DOI: 10.58895/ksp/1000150865 -12 erschienen in:
Forum Bildverarbeitung 2022 | Image Pocessing Forum 2022
DOI: 10.5445/KSP/1000150865 | https://www.ksp.kit.edu/site/books/m/10.58895/ksp/1000150865/

An introduction to quantum image processing on real superconducting quantum computers

Alexander Geng[1,2], Ali Moghiseh[2], Katja Schladitz[2], and Claudia Redenbach[1]

[1] University of Kaiserslautern, Gottlieb-Daimler-Straße 47, 67663 Kaiserslautern
[2] Fraunhofer Institute for Industrial Mathematics ITWM, Fraunhofer-Platz 1, 67663 Kaiserslautern

Abstract The size of images and data we process every day have been growing exponentially over the last years. Quantum computers promise to process this data more efficiently. Experiments on quantum computer simulators prove the paradigms this promise is built on to be correct. However, currently, running the very same algorithms on a real quantum computer is often too error prone to be of any practical use. We explore the current possibilities for image processing on real quantum computers. We redesign a commonly used quantum image encoding technique to reduce its susceptibility to errors. We show experimentally that the current size limit for images to be encoded on the quantum computer and subsequently retrieved with an error of at most 5% is 2×2 pixels. A way to circumvent this limitation is to combine ideas of classical filtering with a quantum algorithm operating locally, only. We show the practicability of this strategy using the application example of edge detection. Our hybrid filtering scheme's quantum part is an artificial neuron, working well on real quantum computers, too.

Keywords Quantum image processing, quantum image encoding, quantum edge detection, quantum artificial neurons, IBM Quantum Experience

1 Introduction

In this contribution, we do not discuss quantum imaging methods. Throughout, we assume the image data to be processed on a quantum computer to be given as a classical gray-value image. Thus, first, we have to encode the gray-value information into quantum states. There are basically three concepts for this encoding, namely basis encoding, phase encoding, and amplitude encoding. Within the last years, several methods have been developed following these three basic concepts [1]. Here, we concentrate on the phase encoding method Flexible Representation of Quantum Images (FRQI) [2] and improve its implementation.

After the encoding, we normally process the states by applying some algorithms. Initially, algorithms were only formulated in theory or executed on simulators of quantum computers. Only since 2016, it has also been possible to execute algorithms on real quantum computers. A short overview of currently available algorithms is given in [3]. Here, we aim at algorithms that run on the actual quantum hardware. More precisely, we implement quantum image processing algorithms on IBM's superconducting quantum computers [4].

This paper is organized as follows. Section 2 provides some basics of quantum computing. In Section 3, we describe the experimental setup including the quantum computers, the software, and the classical computers used. We explain our improved version of the FRQI image encoding in Section 4. In Section 5, we present the idea of hybrid quantum image filtering and highlight the performance for detecting edges in images with a quantum computer. Two variants of the quantum edge detector with 2D and 1D masks are detailed. Section 6 concludes the paper.

2 Quantum computing basics

Before diving into quantum image processing, we summarize some basic concepts of quantum computing [5]. Classical computing and quantum computing follow completely different paradigms, starting with the basic elements. Classically, everything builds on bits, that can attain either state 0 or 1. The quantum analogue are the quantum bits (qubits) – two-state quantum systems that allow for more flexibility.

Analogous to 0 and 1, there are two basis states of a qubit: $|0\rangle = (1,0)^T$ or $|1\rangle = (0,1)^T$. However, any linear combination (superposition)

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{1}$$

of the basis states with $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$ defines a possible state, too. The overall phase of a quantum state is unobservable [5]. That is, $|\psi\rangle$ and $e^{i\xi}\,|\psi\rangle$ for $\xi \in [0, 2\pi]$ define the same state. Hence, it is sufficient to consider $\alpha \in \mathbb{R}$.

As a consequence, the state of a single qubit can be visualized as a point on the unit sphere in $\mathbb{R}^3$ (Bloch sphere) with spherical coordinates ϕ and θ, where $\alpha = \cos(\theta/2)$ and $\beta = e^{i\phi}\sin(\theta/2)$. All operations on a qubit must preserve the condition $|\alpha|^2 + |\beta|^2 = 1$, and can thus be represented by 2×2 unitary matrices. Standard operations (so-called gates) acting on single qubits are

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad P(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}, \tag{2}$$

where the X-gate acts like a classical NOT operator and the Hadamard gate (H) superposes the basic states of a single qubit. A qubit in superposition can be thought of as having all possible states at the same time. The Phase gate (P) rotates by θ about the z-axis of the Bloch sphere. Phase shift gates can be used to encode gray-values.

Additionally, we need operations that link two or more qubits. The most common operation in quantum computing is the controlled NOT-gate (CX-gate) taking two input qubits. The target qubit's state is changed depending on the state of the control qubit:

$$CX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{3}$$

That means, if the control qubit is in state $|1\rangle$, then we apply an X-gate to the target qubit. Otherwise, we do nothing. For example, assume our two qubit system has the state $|10\rangle = |1\rangle \otimes |0\rangle$, where the first qubit is the control, the second the target qubit and $\otimes$ is the tensor product. Then, application of the CX-gate results in the state

$$|11\rangle = |1\rangle \otimes |1\rangle = (0,0,0,1)^T. \tag{4}$$

So basically, the application of quantum gates can be formulated in terms of linear algebra.

In general, we can apply any unitary operation to the target qubit. For example, a controlled-Phase gate applies a P-gate to the target qubit if and only if the control qubit is in state $|1\rangle$. We can also increase the number of control qubits even further. The operation with two control qubits and an X-gate applied to the target qubit is called Toffoli gate.

Applying such controlled operations to two or more qubits with the control qubits in superposition, results in the entanglement of the qubits involved. In terms of linear algebra, an entangled state of several qubits is one that cannot be written as a tensor product of states of the individual qubits. Entanglement is exactly where we benefit from the quantum computing properties. Together with superposition, entanglement allows to use a logarithmically lower number of qubits compared to the number of classical bits.

While in a classical computer all bits are connected to each other, in IBM's quantum computer the qubits are arranged in a special, so-called heavy-hexagonal scheme (see the honeycomb structure in Figure 1). That is, each qubit is directly connected to at most three other qubits. To apply two qubit gates to unconnected qubits, the information has to be swapped to neighbouring qubits by application of additional CX-gates. Each CX-gate, however, increases the overall error considerably such that an algorithm should employ as few CX-gates as possible.

Lastly, the readout is also completely different for classical and quantum computing. On classical computers, you can always read the current state of the bits, copy them, or just continue running an algorithm with the same state of the bits as before the readout. Unfortunately, this is not possible on quantum computers. First, according to the no-cloning theorem [5], a state cannot be copied. Second, when measuring (reading out the state of) a qubit, its state collapses to one of the basis states $|0\rangle$ or $|1\rangle$. Hence, continuing the algorithm after read out is not possible. Additionally, measuring a qubit does not immediately provide the values of α and β in Equation (1). However, the probability of collapsing to $|0\rangle$ is given by $|\alpha|^2$ while the state $|1\rangle$ is obtained with probability $|\beta|^2$. Repeated measurements (shots) of the same state allow for an estimation of these probabilities and thus the values α and β, too. For further reading on quantum computing basics we recommend [5].

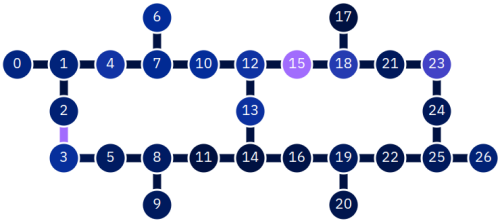

Figure 1: Coupling map of the backends used in this paper. Every circle represents a qubit, lines represent connections between the qubits. Colors code the readout errors (circles) and the CX-errors for the connections (lines). Dark blue indicates a small error, purple a large one. Errors are shown for 'ibmq_ehningen'. 'ibmq_toronto' has the same coupling map, but errors differ slightly (see Table 2).

3 Near-term quantum computers

We use the open-source software development kit Qiskit [6] for working with IBM's circuit-based superconducting quantum computers [4]. They provide a variety of systems, also known as backends, which differ in the type of the processor, the number of qubits (scale), and their connectivity [4]. Access is provided via a cloud. In this paper, we use two of the available 22 backends, 'ibmq_toronto' and 'ibmq_ehningen' see Table 1. This choice is not crucial for our use case as we use a small subset of the qubits only and backends' performance does not differ significantly. The coupling map, so the connections between the qubits, of the backend 'ibmq_ehningen' is shown in Figure 1. Additional parameters describing the performance of IBM's backends are quality (quantum volume) and speed (circuit layer operations per second, CLOPS). All parameters of the two used backends are summarized in Table 1.

Besides the coupling map and the above listed performance values, external conditions influence the backends. Thus, compared to classical computers, the basic operations of quantum computers yield quite large errors. E. g., applying a couple of gates or performing measurements is currently quite noisy with errors that can change hourly. Typical average values for CX error, single qubit gate error, and readout error, are shown in Table 2. Additionally, Table 2 shows the decoher-

Table 1: Processor type and actual performance of the used backends as measured in September 2022.

Backend	Processor type	Scale [# qubits]	Quality [QV]	Speed [CLOPS]
'ibmq_toronto'	Falcon r4	27	32	2.800
'ibmq_ehningen'	Falcon r5	27	64	1.900

Table 2: Typical average calibration data of the two chosen backends. The values are from September 2022.

Backend	CX-error [%]	Single qubit gate error [%]	Readout error [%]	T1 [μs]	T2 [μs]
'ibmq_toronto'	5.34	0.051	3.66	103.71	107.72
'ibmq_ehningen'	0.71	0.024	1.05	151.74	160.92

ence times T1 – a decay constant measuring, how probable a qubit stays in the state $|1\rangle$ and not $|0\rangle$, and T2 – the dephasing time measuring how long the phase of a qubit stays intact. The circuit depth counts the maximal number of basis operations performed by a single qubit during an algorithm. A high circuit depth will result in an accumulation of errors during the runtime of the algorithm.

An additional issue in quantum computing is that only a few operations, called basis gates, can be performed on the quantum computer. Currently, IBM's superconducting quantum computers have five basis gates: the identity, X-, CX-, and P-gates, and the square root X (SX-)gate rotating by $\pi/2$ about the x-axis of the Bloch-sphere [4]. Qiskit includes a transpiler, which decomposes a given algorithm into these basis gates and optimizes these steps in some way [6]. Nevertheless, keeping the available basis gates in mind when developing algorithms helps to limit their overall number.

For preparing data and generating and storing the circuits before sending them to the quantum computer, we use a classical computer with an Intel Xeon E5-2670 processor running at 2.60 GHz, a total RAM of 64 GB, and Red Hat Enterprise Linux 7.9.

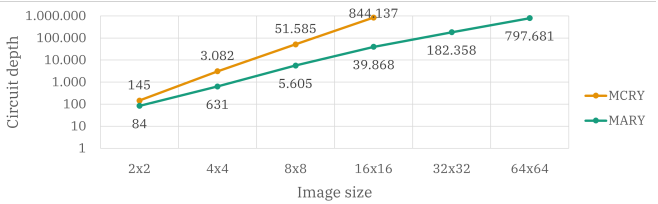

Figure 2: Circuit depth for varying image sizes and MCRY-/MARY-implementation on backend 'ibmq_toronto'. Mean values of 10 observations in logarithmic scale.

4 Quantum image encoding

There are many methods for encoding images in quantum computers. One of the most frequently mentioned methods is FRQI introduced in [2]. Assume that we want to encode a $2^n \times 2^n$ pixel gray-value image. We split the required qubits into two parts - $2n$ qubits for the pixel positions and one qubit for the gray-value information. Practically, FRQI can be implemented on superconducting quantum computers by using entanglement between the position qubits and the gray-value qubit. We take a closer look at the heart of the FRQI algorithm, the multi-controlled y-rotation gate (MCRY). It applies a rotation around the y-axis corresponding to the gray-value only if all position (aka control) qubits are in state $|1\rangle$. Subsequently, we change the state to which the actual phase should be applied by X-gates. Thus,in total we need one MCRY gate for each gray-value in the classical image. As discussed above, on a real backend, complex operations like MCRY have to be constructed by concatenating available basis gates.

Inspired by [7], we replace MCRY by what we call multi-adapted-controlled y-rotation gates (MARY). Our MARY gates need less basis gates, especially less of the particularly error-prone CX-gates. Thus, the replacement reduces the overall error significantly. Moreover, fewer gates and lower circuit depth (Figure 2) speed up calculations. The impact of replacing MCRY by MARY increases with image size. In MCRY, all qubits would ideally have to be connected with each other. Hence, missing connections on the real backends have to be circumvented by swapping with CX-gates. In contrast, MARY requires a much smaller connectivity between the qubits.

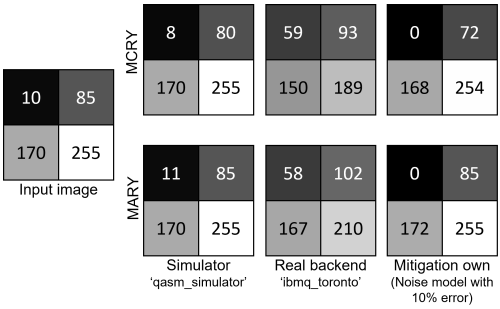

Figure 3: Results for 2×2 gray-value images using the mean of the executions. In the last column, some measurement error mitigation techniques have been applied [8].

Figure 3 shows the performance on a 2×2 sample image. The hardware induced error is clearly visible in the results achieved on the real backend. In fact, there, we can only retrieve the image with acceptable error when applying measurement error mitigation [8]. That is, from observations on exactly this backend, the distribution of the error is estimated. Inversion of the error model then improves the results. To our knowledge, image retrieval with FRQI for images larger than 2×2 is currently not possible on real backends, see also [8–10].

Table 3 shows our findings for the maximum *executable* and *usable* image sizes for the MCRY- and MARY-implementations. Executable here means, it is possible to run the algorithm at all without focusing on the outcomes. Usable implies that the relative difference between input image and reconstructed image is less than 5%. We clearly see a benefit of the MARY-implementation in terms of maximum *executable* image size. However, due to the high noise level of the backends, we could not increase the maximum *usable* image size.

Having experienced this tight restriction, we still aim at image processing algorithms which are robust to the hardware noise in the current noisy intermediate-scale (NISQ) era and hence executable on the real backends. In the next section, we describe a design pattern for algorithms meeting these demands.

Table 3: Current maximum executable and usable image sizes for MCRY- and MARY-implementations on 'qasm_simulator' with 8.192 shots and IBM's backend 'ibmq_toronto' limited to 64 GB memory.

	maximum executable image size		maximum usable image size	
Method	'qasm_simulator'	'ibmq_toronto'	'qasm_simulator'	'ibmq_toronto'
MCRY	32×32	16×16	16×16	2×2
MARY	256×256	32×32	16×16	2×2

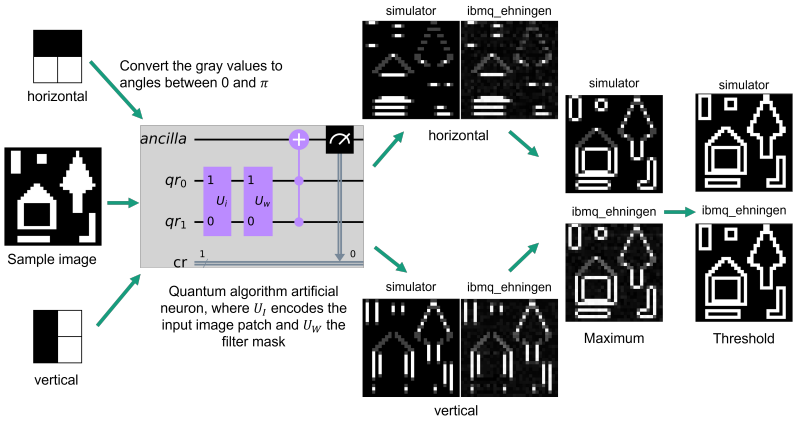

Figure 4: Scheme from [12] for edge detection in a 30×30 sample image by using 2×2 filter masks, 'qasm_simulator' and backend 'ibmq_ehningen' (executed on October, 15 2021) with 8.192 shots, and ToolIP [13] for post-processing.

5 Quantum image filtering

In this section, we introduce a class of hybrid algorithms combining classical filtering with quantum computing on 2 x 2 pixel patches. As an example, we combine classical edge detection with a quantum artificial neuron [11] as sketched in Figure 4. We calculate the inner product of the input image patch and the filter mask not only on a simulator but also on real quantum computers [12]. Being restricted to 2 x 2 masks, we can either apply that directly or split our task into one-dimensional filtering steps. The latter is more robust with respect to noise [12].

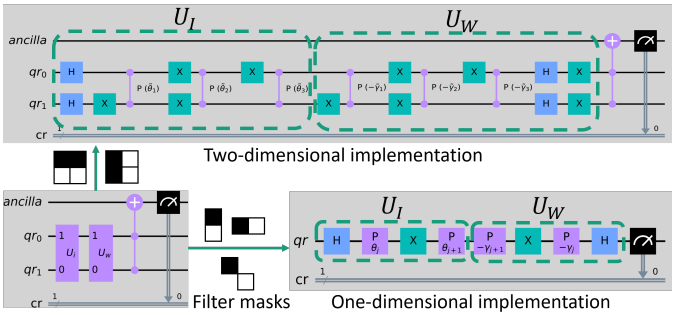

Figure 5: Hybrid quantum edge detection. U_I encodes the input image patch and U_W the filter mask. The gray value information is encoded in the P-gates. In the 1D case, the additional diagonal direction is required for detecting corners, too.

Figure 6: Results for the 256 × 256 House image [14]. The 'qasm_simulator' and backend 'ibmq_ehningen' results differ only slightly.

Moreover, only a very small number of gates and only one qubit per direction and pixel are required. This ensures that a very small number of shots (measurements) suffices for identifying the edges of the image. The lower number of shots in turn reduces the execution time significantly. The quantum circuits of the two implementations are shown in Figure 5.

Figure 6 shows the results of our hybrid 2D edge detection for a typical toy example image [14]. In [12], we process 256 × 256 pixels gray-value images. Further extension to larger images increases the number of circuits, only, but does not decrease the robustness of our algorithm. Nevertheless, in the end, we create one circuit for each combination of

input image patch and filter mask. This can scale up quite fast with larger images. In classical computing, this can be compensated by parallelization. In fact, this is also an option in quantum computing. We can use several qubits in parallel and process multiple image patches at the same time. By that, we decrease the number of needed circuits and also the execution time in the end. Mid-circuit measurement [4] allows to measure a qubit at any step of the algorithm and use the same qubit again for further calculations.

6 Conclusion

Quantum computing is potentially very useful in image processing. It promises exponentially lower memory usage in terms of qubits compared to classical bits and also faster calculations. However, the currently available noisy intermediate-scale quantum computers are still quite error-prone and hardware improvement is subject of vividly ongoing research. At the moment, image retrieval is only possible for images up to a size of 2×2. A strategy to deal with these limitations is to combine quantum and classical algorithms. In such hybrid solutions, the quantum computing part is actually much smaller than the classical part. We use only a small number of gates, and avoid or decrease the number of particularly error-prone types. The quantum computing share can be extended gradually along with the hardware progress. Instead of trying to implement all image processing functionality on quantum computers, we should rather identify, for which problems and which steps in complex algorithms quantum computing can be helpful or eventually even beat classical machines.

Acknowledgement

This work was supported by the project AnQuC-3 of the Competence Center Quantum Computing Rhineland-Palatinate (Germany).

References

1. F. Yan, A. M. Iliyasu, and S. E. Venegas-Andraca, "A survey of quantum image representations," *Quantum Information Processing*, vol. 15, no. 1, pp.

1–35, 2016, https://doi.org/10.1007/s11128-015-1195-6.

2. P. Q. Le, F. Dong, and K. Hirota, "A flexible representation of quantum images for polynomial preparation, image compression, and processing operations," *Quantum Information Processing*, vol. 10, no. 1, pp. 63–84, 2011, https://doi.org/10.1007/s11128-010-0177-y.
3. K. Bharti, A. Cervera-Lierta, T. H. Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J. S. Kottmann, T. Menke *et al.*, "Noisy intermediate-scale quantum algorithms," *Reviews of Modern Physics*, vol. 94, no. 1, p. 015004, 2022.
4. IBM, "IBM Quantum," 2022, https://quantum-computing.ibm.com/. Accessed September 2022.
5. M. A. Nielsen and I. Chuang, *Quantum computation and quantum information*. New York: Cambridge University Press, 2000.
6. H. Abraham *et al.*, "Qiskit: An Open-source Framework for Quantum Computing," 2019, https://doi.org/10.5281/zenodo.2562110.
7. H. Co, E. Peña Tapia, N. Tanetani, J. P. Arias Zapata, and L. García Sanchez-Carnerero, "Quantum imaging processing (a case study: cities at night)," gitHub repository, https://github.com/shedka/citiesatnight. Accessed June 10, 2021.
8. A. Abbas *et al.*, "Learn quantum computation using Qiskit," 2020, https://qiskit.org/textbook/. Accessed September 2022.
9. M. Harding and A. Geetey, "Representation of Quantum Images," 2018, https://www.cs.umd.edu/class/fall2018/cmsc657/projects/group_6.pdf.
10. G. Cavalieri and D. Maio, "A quantum edge detection algorithm," 2020, preprint at https://arxiv.org/abs/2012.11036.
11. S. Mangini, F. Tacchino, D. Gerace, C. Macchiavello, and D. Bajoni, "Quantum computing model of an artificial neuron with continuously valued input data," *Machine Learning: Science and Technology*, vol. 1, no. 4, p. 045008, 2020, https://doi.org/10.1088/2632-2153/abaf98.
12. A. Geng, A. Moghiseh, C. Redenbach, and K. Schladitz, "A hybrid quantum image edge detector for the NISQ era," *Quantum Machine Intelligence*, vol. 4, no. 15, 2022, https://doi.org/10.1007/s42484-022-00071-3.
13. Fraunhofer Institute for Industrial Mathematics, "ToolIP - tool for image processing," www.itwm.fraunhofer.de/toolip. Accessed September 2022.
14. A. Sawchuk *et al.*, "House 4.1.05," USC-SIPI image database, 1973, http://sipi.usc.edu/database/. Accessed September 2022.