Resource Optimal Executable Quantum Circuit Generation Using Approximate Computing

Adarsh, Smaran ; Möller, Matthias

DOI

[10.1109/QCE52317.2021.00040](https://doi.org/10.1109/QCE52317.2021.00040)

Publication date

2021

Document Version

Accepted author manuscript

Published in

2021 IEEE International Conference on Quantum Computing and Engineering (QCE)

Citation (APA)

Adarsh, S., & Möller, M. (2021). Resource Optimal Executable Quantum Circuit Generation Using Approximate Computing. In H. A. Muller, G. Byrd, C. Culhane, & T. Humble (Eds.), *2021 IEEE International Conference on Quantum Computing and Engineering (QCE): Proceedings* (pp. 225-231). [9605317] (Proceedings - 2021 IEEE International Conference on Quantum Computing and Engineering, QCE 2021). IEEE . <https://doi.org/10.1109/QCE52317.2021.00040>

Important note

To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright

Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy

Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

Resource Optimal Executable Quantum Circuit Generation Using Approximate Computing

Smaran Adarsh

Department of Quantum and

Computer Engineering

Delft University of Technology

Delft, The Netherlands

Email: sadarsh.smaranadarsh03@gmail.com

Matthias Möller

Department of Applied Mathematics

Numerical Analysis

Delft University of Technology

Delft, The Netherlands

Email: m.moller@tudelft.nl

Abstract—Quantum Computing is an emerging technology that combines the principles of computer science and quantum mechanics to solve computationally challenging problems significantly faster than classical computers. In this paper, we present a proof-of-principle procedure for generating hardware-executable quantum circuits for Noisy Intermediate-Scale Quantum (NISQ) devices that follows the paradigm of approximate computing.

Our approach starts from the reference circuit and transforms it into an executable circuit with tuneable parameters by replacing the high-level quantum operations by approximate decompositions into hardware-native gates. An inner optimization loop over the rotation gates' angles ensures that the so-created circuit behaves in the same way as the reference one in terms of its expectation-value landscape. This technique is complemented by compiler-based optimizations to further reduce or aggregate gate groups of the optimized circuit. This three-step procedure is embedded into an outer genetic algorithm framework that inspects many different circuit designs with placements of single- and multi-qubit gates according to the hardware's lattice structure, and returns a set of approximate quantum circuits that can be executed on NISQ devices directly.

We have validated our approach for superconducting quantum systems from IBM and Rigetti for various benchmark algorithms. In nearly all cases, our approach outperforms the vendors' quantum-compiler frameworks and produces significantly smaller circuits with up to 50% reduction in the number of gates.

I. BACKGROUND AND PROBLEM STATEMENT

Quantum Computing is an emerging compute technology that has the potential to radically change the way we will be solving computational problems in the future. After decades of theoretical research, the last years have brought the first prototype quantum systems. Though fundamentally different from other computing technologies, quantum computers likewise require a full software stack that enables application developers to express algorithms in a human-readable programming language and transform them into executable instructions.

This is where quantum compiler frameworks come into play. They decompose all software-visible gates into native gates (cf. Table I), map between software and physical qubits, insert additional swap operations to enable multi-qubit operations between non-connected qubits, optimize the circuit to reduce its gate count, and generate electric signals (called control pulses) to implement the circuit on a specific physical device.

Table I
NATIVE GATES OF THE NISQ DEVICES USED IN OUR STUDY

IBM	Rigetti
$R_x(\frac{\pi}{2}), R_z(\lambda), \text{CR}^1$	$R_x(\pm k \frac{\pi}{2}), k \in \mathbb{N}, R_z(\lambda), \text{CZ}, \text{XY}^2$

Various strategies for optimizing circuit properties have been proposed in the literature ranging from rule-based substitution and compiler-based optimization that exploits the commutativity of gate groups and performs instruction aggregation [1] to machine learning for creating surrogate circuits via a training process [2–5]. Yet another approach consists in recompiling a quantum circuit given in term of its unitary matrix U into a surrogate unitary V with adjustable parameters using the overlapping of the matrices, $\text{Tr}(V^*U)$, as cost function [6, 7].

In this paper, we propose a novel multi-level optimization approach that builds on the paradigm of approximate computing. In particular, we consider the placement of native quantum gates that can be physically implemented on existing quantum systems (IBM, Rigetti) as *circuit topology optimization problem* and the adjustment of parameters of these native quantum gates as *continuous parameter optimization problem*. The outer topology problem is solved with the aid of a genetic algorithm, whereas the gradient-descent method is used for the inner optimization. The latter is complemented by *compiler-based optimizations*, e.g., to aggregate gate groups.

Despite the common use of continuous optimization techniques, our approach towards generating hardware-optimized quantum circuits is fundamentally different from the ones presented in [2–5] whose core idea is to adopt generic surrogate circuits, built up from the native gate set and possibly taking into account the device-specific qubit connectivity, and to tune them to mimic the behaviour of a certain target circuit.

In contrast, our approach starts from the reference circuit and transforms it into an executable one by applying rule-based *approximate* gate substitutions whose parameters are

¹The cross resonance (CR) gate is considered to be the *CNOT* quantum gate according to the implementation on the IBM hardware via Qiskit [8].

²The parameterized ISWAP (XY) gate has been introduced recently with PyQuil [9] version 2.22.0 and is not yet considered in our multi-level circuit optimization approach.

optimized afterwards. This very much simplifies the task of the continuous optimizer since even the initial guess of the so-created circuit inherits some basic structure from the reference one. Moreover, our approach immediately yields an upper bound for the worst-case scenario, that is, the circuit with the highest gate count, which is the primary objective quantity that needs to be minimized in practice, at least for NISQ devices.

This combination of rule-based approximate substitution and parameter optimization makes it possible to fine-tune the resulting circuit in various ways. Firstly, the circuit can be tuned to perform ‘as good as possible’ in terms of its expectation-value landscape within the resource limitations imposed by today’s quantum hardware. Secondly, it is possible to fine-tune the circuit’s expectation-value behaviour for a small subset of all possible input states that are relevant for a particular application. Finally, the level of exactness, i.e. the cost function tolerance, can be used as parameter to control the costs of the circuit generation procedure. That is, the user can choose between the fast generation of inaccurate and the more time-consuming generation of more accurate circuits.

The rest of this paper is organized as follows: Section II presents our multi-level optimization approach in more detail followed by its experimental evaluation for a collection of quantum circuits in comparison with quantum compiler toolchains provided by IBM and Rigetti in Section III. Conclusions are drawn in Section IV.

II. MULTI-LEVEL OPTIMIZATION APPROACH

Let $\mathcal{B}$ denote the computational basis of an n -qubit system and Ψ the infinite set of all admissible quantum states for an n -qubit register $|\psi\rangle$. Furthermore, let the reference circuit be represented as unitary $2^n \times 2^n$ matrix U , which acts as the mapping between input and output states, i.e. $|\psi_{\text{out}}\rangle = U|\psi_{\text{in}}\rangle$. In practice, it suffices to consider a finite subset of Ψ , e.g., the 2^n basis states and some linear combinations thereof, and study the action of U on each of them individually.

The overall aim of our multi-level optimization approach is to find an *approximate circuit* that consists of native gates only, has minimal gate count and mimics the behaviour of the reference circuit as good as possible. An example for the 2-qubit *controlled-NOT* (*CNOT*) gate [10] is given in Figure 1, whose derivation will be detailed in Sec. II-A below. Our long-term goal is to also take into account the qubit connectivity of real quantum processors and the complex error behaviour of specific physical quantum devices.

Let $\theta_{\hat{U}} = (\theta_1, \dots, \theta_N)^\top$ be the vector of parameters for the approximate circuit $\hat{U}$ and let $\mathcal{U}_V$ denote the set of admissible approximate circuits with fixed size V . The synthesis process can then be written as a global optimization problem:

$$\hat{U}_{\text{opt}} = \underset{\hat{U} \in \mathcal{U}_V}{\text{argmin}} \min_{\theta_{\hat{U}}} \max_{|\psi\rangle \in \Psi} F(|\psi\rangle; U, \hat{U}, \theta_{\hat{U}}), V \rightarrow \min, \quad (1)$$

where the parameterized cost function is defined as follows

$$F(|\psi\rangle; U, \hat{U}, \theta_{\hat{U}}) = \sum_{k=1}^n \left(\langle A^k \rangle_{U|\psi\rangle} - \langle A^k \rangle_{\hat{U}(\theta_{\hat{U}})|\psi\rangle} \right)^2. \quad (2)$$

Figure 1. Decomposition of the 2-qubit *CNOT* gate into native gates ($\theta_1 = \theta_2 = -\frac{\pi}{2}$ and $\theta_3 = \theta_4 = \frac{\pi}{2}$)

Here, A^k is an observable, e.g., Pauli-X, -Y, or -Z gate, applied to the k^{th} qubit. Furthermore, the expectation value [11] in the state $|\psi\rangle$ upon application of the operator P is denoted by

$$\langle A^k \rangle_{P|\psi} = \langle (P|\psi)^\dagger | A^k | P|\psi \rangle; \quad (3)$$

The cost function equals zero when the approximate circuit mimics its reference counterpart *exactly* for all states in Ψ . Our goal is to create *approximate* circuits with a small gate count (fewer resources), for which the value of the cost function drops below a user-defined threshold for all input states.

(1) is solved numerically using the multi-level optimization algorithm sketched in Figure 2, whose different algorithmic components will be addressed in the forthcoming subsections.

Figure 2. Flow chart of the multi-level optimization algorithm. The reiteration from circuit optimizer to RBAGS (dashed arrow) is not yet included in our proof-of-concept implementation but supported conceptually

A. Rule-based approximate gate substitution (RBAGS)

The first step of our algorithm consists of replacing all software-visible gates with approximate gate decompositions. Here and below we use the *CNOT* gate depicted in Figure 1 as an example to illustrate the steps of our algorithm. Its decomposition into two *Hadamard* (*H*) gates with an adjacent *controlled-Z* (*CZ*) gate is common practice [10] as is the substitution of *H* by one of the following rules [12]

$$\begin{aligned} H &\mapsto R_x(\pi) R_y\left(\frac{\pi}{2}\right), & H &\mapsto R_y\left(-\frac{\pi}{2}\right) R_x(\pi), \\ H &\mapsto R_z(\pi) R_y\left(-\frac{\pi}{2}\right), & H &\mapsto R_y\left(\frac{\pi}{2}\right) R_z(\pi). \end{aligned}$$

The problem of the above *exact* decompositions is that the angles are fixed and leave no room for optimization by adjusting parameters. Closer inspection of, say, the first substitution rule reveals that the application of $R_y\left(\frac{\pi}{2}\right)$ alone suffices to

take $|0\rangle$ and $|1\rangle$ into the superposition states $|+\rangle$ and $|-\rangle$ [12], respectively. $R_y(\frac{\pi}{2})$ and $R_x(\pi)$ together take care of the unitary and Hermitian property of the *Hadamard* gate.

It turns out that the following four *approximate* decompositions often suffice to mimic the behaviour of the *Hadamard* gate in practice and with the available native gate sets:

$$\begin{aligned} L0 : H &\mapsto R_x(\theta), & L1 : H &\mapsto R_y(\theta), \\ L2 : H &\mapsto R_x(\theta_1)R_z(\theta_2), & L3 : H &\mapsto R_z(\theta_1)R_x(\theta_2). \end{aligned}$$

For convenience, let us number the two *Hadamard* gates in Figure 1 from left to right. As will be demonstrated in the sequel, the choice [3, 2], that is the first *H*-gate is substituted by the approximate gate decomposition labeled *L3*, the second *H*-gate by *L2*, leads to a valid circuit, whereas [0, 1] and [1, 0] do not. This is not a problem since inappropriate circuits will be detected by the large values of the cost function (2) and rejected by the topology optimization algorithm; Sec. II-D.

RBAGS moreover takes into account *generic* quantum gate identities [13] valid for arbitrary rotation angles, e.g.

$$HR_x(\theta)H = R_z(\theta), \quad 0 \leq \theta \leq 2\pi \quad (4)$$

to reduce the circuit's gate count. An exhaustive search over *all* possible substitutions yields 4^2 combinations for a simple *CNOT* gate and becomes prohibitively expensive for realistic quantum circuits. A genetic algorithm will be discussed in Sec II-D to replace exhaustive searches in practice. For the next subsections, we restrict ourselves to a single circuit $\hat{U}$ with parameters $\theta_{\hat{U}}$. Also, we will continue the description with the focus on approximating *Hadamard* gates.

B. Continuous parameter optimization

In the second step, the circuit's parameters θ_j are optimized until the cost function (2) is sufficiently small. Let $\hat{U}(\theta)$ denote the approximate circuit depicted in Figure 1 with the four tuning parameters $\theta = (\theta_1, \theta_2, \theta_3, \theta_4)^\top$. Here and below, subscript $\hat{U}$ has been dropped for the sake of better readability.

Starting with either randomly initialized data or values close to the natively supported angles (e.g., $R_x(\pm k\frac{\pi}{2})$ for $k \in \mathbb{N}$ for Rigetti) the parameter vector is updated successively using a stochastic gradient-descent optimizer of the form

$$\theta^{t+1} = \theta^t - \gamma \nabla_{\theta} F(|\psi\rangle; U, \hat{U}, \theta^t). \quad (5)$$

Here, γ is the user-defined step size and ∇_{θ} denotes the gradient of the cost function (2) with respect to the tuneable rotation angles. The optimizer takes into account *all* $|\psi\rangle \in \Psi$ simultaneously and returns a parameter vector that is optimal for *all* input states under consideration at the same time.

We use the PennyLane library [14], which allows to differentiate quantum circuits automatically and provides optimization and machine learning tools to determine the optimal design parameters for a finite set of input states $|\psi\rangle$ efficiently. The iteration procedure (5) is stopped once the cost function value has dropped below a user-defined tolerance. For the example circuit depicted in Figure 1, these are $\theta_{\text{opt}} = (-\frac{\pi}{2}, -\frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2})^\top$ thereby turning it into an exact decomposition of the *CNOT*.

C. Compiler-based circuit optimization

By nature of continuous optimization, the design parameters are not given as symbolic values but as floating-point numbers. To carry out the compiler-based optimization steps described in the following, numerical data can either be converted to symbolic values or handled with a safety interval, e.g. all values x with $|x - \pi| < \text{tol}$ are identified with the symbol π . With this convention in mind, our compiler-based circuit optimization performs the following steps:

R1: Collapsing of adjacent gates considering

(a) cancellation of inverse gates, e.g.

$$CZCZ^\dagger = \text{id}, \quad R_x(\theta)R_x(-\theta) = \text{id}$$

(b) fusion of rotation gates, e.g.

$$R_z(\theta_1)R_z(\theta_2) = R_z(\theta_1 + \theta_2)$$

(c) aggregation using commutativity rules [15]

R2: Substitution of *specific* gate identities [13], e.g.

$$HR_z(\theta)H = R_x(\theta), \quad \theta = \pi, \pm\frac{\pi}{2}$$

R3: Elimination of rotation gates whose design parameters have not changed compared to the initial values as their presence does not affect the circuit behaviour.

The third rule, R3, is optional and requires some explanation. Obviously, a gate, say, $R_x(\pi/2)$ might be relevant for the correct functioning of the circuit while at the same time R3 would eliminate it if its rotation angle had been initialized to $\pi/2$ from the outset. In our implementation we therefore initialize the rotation angles with values “close to” the expected values, e.g., $R_x(1.5)$. This, on the one hand, speeds up the convergence of the optimization process [16] and, on the other hand, ensures that some variation of the initial rotation angles occur unless the gate is not required and can be safely eliminated by the compiler-based optimization pass.

Any of the above modifications might enable further optimizations with adjacent gates not possible in previous rounds so that multiple sweeps are performed until no further changes are observed. In our experience, compiler-based optimization leads to a significant reduction of the circuit's depth and gate count. Figure 3 illustrates how multiple compiler sweeps reduce the depth and the overall size for *CNOT* to 3 by swapping commuting gates and cancelling inverse ones afterwards.

D. Circuit topology optimization

Obviously, exhaustive search over all possible gate substitutions in the first step of our algorithm is only possible for very small circuits. Already a moderate-size circuit with 100 *H* gates would require an exhaustive search over 4^{100} variations, which is impossible to solve by a brute-force approach. As a remedy, we resort to a genetic algorithm [17] approach, which adopts a meta-heuristic strategy inspired by natural selection to efficiently explore the search space in rounds.

In the beginning, a subset of all possible circuits $\hat{U}$ is selected randomly and declared generation zero. Each of these circuits is optimized independently and at best in parallel using

Figure 3. Hardware-optimized approximate circuit for the 2-qubit *CNOT* gate after exploiting (a) the commuting property of $R_z(-\pi/2)$ and *CZ* and (b) cancellation of adjacent R_z gates with polar opposite angles $\pm\pi/2$

the workflow described in Sections II-A–II-C. The values of the cost function (2) evaluated for the fully optimized circuits serve as a criterion to measure the circuits’ fitness. A few fittest circuits are carried over to the next generation without modification to secure the reached fitness level for all times. The rest of the next generation is created by performing crossover [18] and mutation [19] operation on the fittest circuits. Multiple rounds of this evolutionary algorithm are performed until no further improvements are obtained or the maximum number of iterations has been reached. From the final generation, the circuit with the lowest gate count that also has a cost function value below the user-specified threshold is selected as the global optimum to problem (1).

Figure 4. Grover’s algorithm for two qubits [20]

Let us consider Grover’s algorithm [20] for 2 qubits as depicted in Figure 4, whose 6 *H*-gates give rise to 4^6 possible topology configurations. The oracle can be interpreted as a user-definable callback function that will give rise to additional gates, which are, however, not of Hadamard type and are therefore tacitly ignored in what follows.

Our initial population consists of 1024 approximate circuits that are created from the reference circuit by randomly applying one of the four substitution rules $L0 - L3$ per *H*-gate. By numbering the 6 *H*-gates from top left to bottom right, this selection can be easily encoded as integer vector, e.g., $[3, 1, 2, 3, 0, 1]$, which makes the implementation of *crossover* and *mutation* operations straightforward as shown in the two Algorithms 1 and 2, respectively.

Example 1. Let $P_1 = [3, 1, 2, 3, 0, 1]$ and $P_2 = [2, 3, 0, 1, 1, 0]$ be a pair of fit circuits from the parent generation and denote the two randomly generated masks by $M_1 = [0, 1, 1, 0, 0, 1]$ and $M_2 = [1, 0, 1, 0, 1, 0]$, respectively. The two offsprings generated by Algorithm 1 are then given by $O_1 = [3, 3, 0, 3, 0, 0]$ and $O_2 = [3, 3, 2, 1, 0, 0]$, whereby “?” denotes the ternary if-then-else operator.

The mutation acts on a single parent circuit and creates two new offsprings by permuting the index vector randomly and reversing the order of substitutions, respectively. These mutation operations are meant to keep the population’s diversity and enhance the chances of better exploring the search space.

Algorithm 1: Crossover Operation

```

for Every  $(P_1, P_2)$  of fit circuits from the parent
  generation do
    Create random masks  $M_1, M_2$  ; // Bitstring of
    length 6.
    for Every bit in masks do
      /* First offspring */
       $O_1[i] = (M_1[i] == 0 ? P_1[i] : P_2[i]);$ 
      /* Second offspring */
       $O_2[i] = (M_2[i] == 0 ? P_2[i] : P_1[i]);$ 
    end
  end

```

Algorithm 2: Mutation Operation

```

for Every fit circuit  $P$  from the parent generation do
  /* First offspring */
   $O_1 = \text{RandPermute}(P);$ 
  /* Second offspring */
   $O_2 = \text{Reverse}(P);$ 
end

```

Example 2. Let $P = [3, 1, 2, 3, 0, 1]$ denote a fit circuit from the parent generation. A possible first offspring resulting from applying Algorithm 2 is given by $O_1 = [3, 3, 1, 1, 0, 2]$, where the second offspring equals $O_2 = [1, 0, 3, 2, 1, 3]$.

III. EXPERIMENTAL EVALUATION

In this section we apply our multi-level optimization approach to a collection of software-visible multi-qubit gates and quantum circuits to evaluate its performance in comparison with two state-of-the-art quantum compiler toolchains implemented in IBM’s Qiskit framework [8] and Rigetti’s pyQuil library [9] that are used regularly for production runs. All tests were conducted on a consumer laptop with Intel Core i7-7700HQ Quad Core Processor and 8GB GDDR5 RAM based on a prototype implementation of our approach in the PennyLane library [14]. The specific versions of the software libraries and toolchains used in our study are as follows:

- PennyLane version 0.14.0,
- PyQuil / Forest SDK version 2.20.0,
- Qiskit version 0.23.2.

IBM’s `qiskit.compiler.transpile` function offers three optimization levels controlled by the parameter `optimization_level`. Level-1 optimization collapses adjacent gates that cancel each other. Level-2 optimization does

Table II
PERFORMANCE EVALUATION OF OUR APPROACH FOR QUANTUM
BENCHMARK CIRCUITS GENERATED FOR THE RIGETTI PLATFORM

Quantum circuit	No. of qubits	No. of quantum gates	
		Rigetti	Ours
CNOT	2	6 (50.00%)	3
SWAP	2	14 (35.71%)	9
QFT	2	18 (11.12%)	16
QFT	5	118 (22.03%)	92
Toffoli	3	33 (54.54%)	15
4-qubit Quantum Adder	8	197 (33.00%)	132
8-qubit Quantum Adder	16	474 (34.17%)	312
MC3X	4	90 (15.55%)	76
MC4X	5	195 (15.89%)	164
Grover (excl. oracle)	2	15 (46.66%)	8
BV (excl. oracle)	4	21 (47.62%)	11

inverse gate cancellation as the former level and, in addition, exploits commutativity properties to enable further adjacent gate cancellations. Level-3 optimization performs all level-2 optimizations and, additionally, re-synthesizes sub-blocks of the quantum circuit to aggregate over sets of gate combinations. Since IBM’s most aggressive level-3 optimizer is not deterministic, 500 runs were performed for each benchmark from which the best results are reported in Table III and compared to the outcome of our circuit generation approach.

In contrast to the multiple optimization levels offered by the IBM transpiler, Rigetti’s compiler, which is invoked by the function `qc.compiler.quil_to_native_quil`, has a single optimization level, which collectively includes the optimizations considered by IBM’s transpiler up to a certain extent. The experimental results are given in Table II.

In both cases, we adjusted our multi-level optimization approach – mainly the set of approximate substitution and gate group aggregation rules – to use the native gate sets of the platform at hand as given in Table I. Moreover, since our approach does not yet account for device-specific lattice structures, the vendors’ compiler toolchains were invoked assuming perfect qubits and full qubit connectivity.

A. Quantum circuits without oracles

The benchmarks considered in our experimental study fall into one of the following three categories: *Controlled-NOT* [10], *SWAP* [21], *Toffoli* [22], [23] and the *Multi-Controlled X-gates* are basic multi-qubit quantum gates that are widely used in larger quantum circuits. The *Quantum Fourier Transform (QFT)* [24] and the *Quantum Adders* [25] are examples of stand-alone quantum algorithms. Finally, we consider two well-known quantum algorithms, *Grover’s search* [20] and the *Bernstein-Vazirani (BV) algorithm* [26], which make use of oracle functions whose treatment will be detailed in Sec. III-B.

In general, the quantum gate counts presented in Tables II and III indicate that our multi-level optimization approach is able to reduce the number of native gates significantly and, hence, lead to smaller executable circuits, in comparison with the optimized circuits produced by the vendors’ toolchains.

Table III
PERFORMANCE EVALUATION OF OUR APPROACH FOR QUANTUM
BENCHMARK CIRCUITS GENERATED FOR THE IBM PLATFORM

Quantum circuit	No. of qubits	Optimization	No. of quantum gates	
			IBM	Ours
QFT	2	Level 1	12 (8.33%)	11
		Level 2	12 (8.33%)	
		Level 3	42 (73.80%)	
QFT	5	Level 1	57 (1.75%)	56
		Level 2	57 (1.75%)	
		Level 3	73 (23.28%)	
Toffoli	3	Level 1	18 (44.44%)	10
		Level 2	18 (44.44%)	
		Level 3	18 (44.44%)	
4-qubit Quantum adder	8	Level 1	122 (39.34%)	74
		Level 2	116 (36.20%)	
		Level 3	131 (43.51%)	
8-qubit Quantum adder	16	Level 1	286 (39.16%)	174
		Level 2	272 (36.02%)	
		Level 3	299 (41.80%)	
MC3X	4	Level 1	46 (21.73%)	36
		Level 2	46 (21.73%)	
		Level 3	102 (64.70%)	
MC4X	5	Level 1	94 (22.92%)	76
		Level 2	94 (22.92%)	
		Level 3	150 (49.33%)	
Grover (excl. oracle)	2	Level 1	17 (52.94%)	8
		Level 2	16 (50.00%)	
		Level 3	27 (70.37%)	
BV (excl. oracle)	4	Level 1	22 (50.00%)	11
		Level 2	22 (50.00%)	
		Level 3	22 (50.00%)	

The specific gains, however, largely depend on the concrete circuit, its gates and the device-specific native gate set.

As an example, consider the two QFT circuits for 2 and 5 qubits, respectively. While our approach leads to a 10-20% reduction in the gate count for Rigetti the savings on the IBM platform are only marginal, namely, 1.75%, which is due to the difference in the native gate sets, in particular, the availability of *CNOT* as native gate on IBM. On the other hand, this evaluation demonstrates that our multi-level optimization approach is universal in the sense that it can easily be ported to other native gate sets by adjusting the approximate gate substitution and gate group aggregation rules.

B. Quantum circuits with oracles

In contrast to the benchmarks from the previous section, the quantum algorithms considered here involve oracle functions, which can be considered as user-definable callbacks.

Before proceeding to the experimental evaluation of *Grover’s search algorithm* [20] and the *Bernstein-Vazirani algorithm* [26] we need to address a fundamental difference between our multi-level optimization approach and the quantum compiler frameworks from IBM and Rigetti. By optimizing the approximate circuit with respect to the cost function (2), our algorithm ensures that the expectation values of the optimized $\hat{U}(\theta_{\hat{U}})$ and the reference circuit U evaluated in the computational basis are as close as possible.

This, however, *does not* imply that $U|\psi\rangle = \hat{U}(\theta_{\hat{U}})|\psi\rangle$ for all $|\psi\rangle \in \Psi$. Hence, our approach cannot be used to optimize multiple sub-circuits, say, $\hat{U}_1(\theta_{\hat{U}_1})$ and $\hat{U}_2(\theta_{\hat{U}_2})$ individually and pieced them together to a larger circuit as follows $\hat{U}(\theta_{\hat{U}_1}, \theta_{\hat{U}_2}) := \hat{U}_2(\theta_{\hat{U}_2}) \circ \hat{U}_1(\theta_{\hat{U}_1})$. Instead, the so-composed circuit needs to be re-optimized *as a whole* to

guarantee correct functioning, whereby the re-optimization process largely benefits from the good initial guess of rotation angles. What *does* work is to prepare the input state with an abstract preparatory circuit that is tacitly assumed to be given in terms of native gates and feed its output as input into an approximate circuit that has been optimized stand-alone.

For both oracle-based benchmarks we have considered a fixed oracle function but safeguarded it with barriers to exclude it from optimizations such as gate cancellation and aggregation. Moreover, we excluded the number of gates required for the oracle function in the gate counts reported in Tables II and III. For both quantum computing platforms, our approach leads yields a significant reduction of gates by roughly 50%.

C. Preliminary results obtained on real quantum hardware

Let us conclude the experimental evaluation of our approach by studying the behaviour of an approximate circuit for the 3-qubit *Toffoli* gate (*CCNOT*) on IBM’s 5-qubit quantum device *ibmq_5_yorktown*. We have chosen this relatively old device (Canary r1 processor, quantum volume of 8) as its lattice structures consists of two 3-cycle graphs, which makes it possible to execute 3-qubit circuits without the need to introduce additional *SWAP* gates as overhead.

Figure 5 depicts the probabilities of measuring the expected outcome for 20 independent runs with 2048 shots per run. Obviously, our approximate circuit is significantly and consistently more reliable than the one generated by the IBM transpiler with level-3 optimization. This is due to the significant reduction of *CNOT* gates by 33.33% (6 *CNOT* gates in the IBM-generated circuit vs. 4 *CNOT* gates in our approximate circuit) and the overall reduction of gates by 44%.

It should be noted that our approach does not implement any mechanism to actively reduce the number of *CNOT* gates as it is the case for instance in [7]. In our case, the reduction of *CNOT* gates is the result of *approximate* gate substitutions that enable more aggressive gate cancellation and agglomeration, which thereby also affects the number of *CNOT* gates.

Figure 5. Noise reduction for the 3-qubit Toffoli gate.

A similar trend can be observed in Figure 6, which depicts the probabilities of measuring the correct output of a 2-qubit *Quantum Adder*. This circuit requires 4 qubits in total and,

Figure 6. Noise reduction for the 2-qubit Quantum Adder

hence, additional *SWAP* operations on the Yorktown processor. Despite this overhead, our approximate circuit is still less susceptible to errors compared to the exact circuit produced by IBM’s transpiler. This is, again, due to the significant, not actively enforced reduction of *CNOT* gates by 25% (16 *CNOT* gates in the IBM-generated circuit vs. 12 *CNOT* gates in our approximate circuit) and the overall reduction of gates by 40%.

On an additional note, for both circuits (*Toffoli* and *Quantum Adder*), the probability of measuring the expected outcome is not close to 1 since both circuits implement a large number of error-prone *CNOT* operations. A general strategy for improving the fidelity of approximate circuits is to focus on reducing the number of error-prone gates possibly at the cost of allowing slightly more but less error-prone gates. This as well as other optimization strategies will be investigated in future works.

The experimental evaluation of the larger circuits from Table III is currently underway and will be presented in a forthcoming publication. The main challenge is the lack of control on the placement of logical qubits on their hardware counterparts and the insertion of additional *SWAP* gates, which we leave to IBM’s transpiler toolchain for the time being.

IV. CONCLUSIONS AND FUTURE WORK

We have presented a multi-level optimization approach to generate hardware-optimized quantum circuits from textbook quantum algorithms in a fully automated workflow and compared its performance to that of the state-of-the-art quantum compiler toolchains from IBM and Rigetti.

In all benchmarks considered in our study, our approach outperforms the established tools or is on par with them concerning the capability to reduce the circuits’ gate count and increase their fidelity when executed on NISQ devices from IBM. Our current proof-of-concept implementation mainly serves as technology demonstrator of the new circuit-generation approach that builds on the paradigm of approximate computing and is not yet tuned with respect to computational performance nor can it replace the general-purpose toolchains from IBM or Rigetti as it can only handle *Hadamard* gates so far. We would like to stress the fact that

although our approach is based on approximate gate substitution, the resulting circuits perform the exact functionality from a hardware perspective as their textbook counterparts.

Our future work will focus on speeding up the computational workflow by parallelizing the genetic algorithm on distributed computer systems and on extending the approximate substitution rules and the expressions for the compiler-based optimization sweeps to support a wider range of software-visible gates. Additionally, we will integrate noise models into our cost function to generate approximate circuits that mitigate the imperfect behavior of physical quantum systems.

We finally plan to take the qubit connectivity of real quantum processors into account by supporting extra qubit permutations between gates. This and the above features are all conceptually supported by our multi-level optimization framework making it a powerful tool for the automated generation of approximate quantum circuits for NISQ devices.

REFERENCES

- [1] Y. Shi, N. Leung, P. Gokhale, Z. Rossi, D. I. Schuster, H. Hoffmann, and F. T. Chong, "Optimized compilation of aggregated instructions for realistic quantum computers," ser. ASPLOS '19, 2019. [Online]. Available: <https://doi.org/10.1145/3297858.3304018>
- [2] M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini, "Parameterized quantum circuits as machine learning models," *QST*, 2019. [Online]. Available: <https://doi.org/10.1088/2058-9565/ab4eb5>
- [3] M. Benedetti, D. Garcia-Pintos, O. Perdomo, V. Leyton-Ortega, Y. Nam, and A. Perdomo-Ortiz, "A generative modeling approach for benchmarking and training shallow quantum circuits," *npj Quantum Information*, 2019. [Online]. Available: <https://doi.org/10.1038/s41534-019-0157-8>
- [4] S. Khatri, R. LaRose, A. Poremba, L. Cincio, A. T. Sornborger, and P. J. Coles, "Quantum-assisted quantum compiling," *Quantum*, 2019. [Online]. Available: <https://doi.org/10.22331/q-2019-05-13-140>
- [5] D. Zhu, N. M. Linke, M. Benedetti, K. A. Landsman, N. H. Nguyen, C. H. Alderete, A. Perdomo-Ortiz, N. Korda, A. Garfoot, C. Brecque, L. Egan, O. Perdomo, and C. Monroe, "Training of quantum circuits on a hybrid quantum computer," *Science Advances*, 2019. [Online]. Available: <https://advances.sciencemag.org/content/5/10/eaaw9918>
- [6] T. Jones and S. C. Benjamin, "Quantum compilation and circuit optimisation via energy dissipation," *arXiv preprint arXiv:1811.03147*, 2018.
- [7] M. G. Davis, E. Smith, A. Tudor, K. Sen, I. Siddiqi, and C. Iancu, "Heuristics for quantum compiling with a continuous gate set," *arXiv preprint arXiv:1912.02727*, 2019.
- [8] H. Abraham and et al., "Qiskit: An open-source framework for quantum computing," 2019.
- [9] R. S. Smith, M. J. Curtis, and W. J. Zeng, "A practical quantum instruction set architecture," 2016.
- [10] J. C. Garcia-Escartin and P. Chamorro-Posada, "Equivalent quantum circuits," 2011.
- [11] P. M. Bakshi and K. T. Mahanthappa, "Expectation value formalism in quantum field theory. i," *Journal of Mathematical Physics*, 1963.
- [12] G. E. Crooks, "Gates, states, and circuits," 2020.
- [13] C. Lomont, "Quantum circuit identities," *arXiv preprint quant-ph/0307111*, 2003.
- [14] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, M. S. Alam, S. Ahmed, J. M. Arrazola, C. Blank, A. Delgado, S. Jahangiri, K. McKiernan, J. J. Meyer, Z. Niu, A. Száva, and N. Killoran, "PennyLane: Automatic differentiation of hybrid quantum-classical computations," 2020.
- [15] G. G. Guerreschi and J. Park, "Gate scheduling for quantum algorithms," *ArXiv e-prints*, 2017.
- [16] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *ICML*. PMLR, 2013.
- [17] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, 1994.
- [18] A. J. Umbarkar and P. D. Sheth, "Crossover operators in genetic algorithms: a review." *IJSC*, 2015.
- [19] O. Abdoun, C. Tajani, and J. Abouchabka, "Hybridizing psm and rsm operator for solving np-complete problems: Application to travelling salesman problem," *arXiv preprint arXiv:1203.5028*, 2012.
- [20] L. K. Grover, "A fast quantum mechanical algorithm for database search," ser. STOC '96. ACM, 1996. [Online]. Available: <https://doi.org/10.1145/237814.237866>
- [21] J. C. Garcia-Escartin and P. Chamorro-Posada, "A swap gate for qudits," *QIP*, 2013.
- [22] E. Fredkin and T. Toffoli, "Conservative logic," *Int J Theor Phys*, 1982.
- [23] V. V. Shende and I. L. Markov, "On the cnot-cost of toffoli gates," *arXiv preprint arXiv:0803.2316*, 2008.
- [24] L. Ruiz-Perez and J. C. Garcia-Escartin, "Quantum arithmetic with the quantum fourier transform," *QIP*, 2017.
- [25] H. Thapliyal and N. Ranganathan, "Design of efficient reversible logic-based binary and bcd adder circuits," *JETC*, 2013.
- [26] E. Bernstein and U. Vazirani, "Quantum complexity theory," *SICOMP*, 1997.