Quantum Gate Decomposition

A Study of Compilation Time vs. Execution Time Trade-offs

Evandro C. R. Rosa
Departamento de Informática e Estatística,
Universidade Federal de Santa Catarina
Florianópolis, Brazil
evandro.crr@posgrad.ufsc.br

Jerusa Marchi
Departamento de Informática e Estatística,
Universidade Federal de Santa Catarina
Florianópolis, Brazil
jerusa.marchi@ufsc.br

Eduardo I. Duzzioni
Departamento de Física,
Universidade Federal de Santa Catarina
Florianópolis, Brazil
eduardo.duzzioni@ufsc.br

Rafael de Santiago
Departamento de Informática e Estatística,
Universidade Federal de Santa Catarina
Florianópolis, Brazil
r.santiago@ufsc.br

ABSTRACT

Similar to classical programming, high-level quantum programming languages generate code that cannot be executed directly by quantum hardware and must be compiled. However, unlike classical code, quantum programs must be compiled before each execution, making the trade-off between compilation time and execution time particularly significant. In this paper, we address the first step of quantum compilation: multi-qubit gate decomposition. We analyze the trade-offs of state-of-the-art decomposition algorithms by implementing them in the Ket quantum programming platform and collecting numerical performance data. This is the first study to both implement and analyze the current state-of-the-art decomposition methods within a single platform. Based on our findings, we propose two compilation profiles: one optimized for minimizing compilation time and another for minimizing quantum execution time. Our results provide valuable insights for both quantum compiler developers and quantum programmers, helping them make informed decisions about gate decomposition strategies and their impact on overall performance.

KEYWORDS

Quantum Computing, Quantum Programming, Quantum Compiler, Gate Decomposition, Ket

1 Introduction

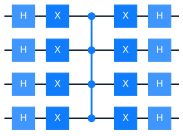
For programming languages and platforms that treat the quantum bit (qubit) as a first-class citizen in quantum programming, *e.g.*, Q# [26] and Ket [7], coding a quantum application involves invoking functions known as quantum gates. Quantum gates have no side effects on the classical state of the program; they only affect the quantum state and can induce superposition and entanglement [19]. Another class of functions, known as measurements, is used to extract information from the quantum state, returning it as classical data while causing the collapse of the quantum state, *i.e.*, destroying the superposition.

Figure 1a illustrates a simple quantum program written in Python using the Ket quantum programming platform [7]. This code implements the Grover diffusion operator, a key component of Grover's quantum search algorithm [12], utilizing the Hadamard (H), Pauli X, and Pauli Z gates. The term *quantum gate*, or more precisely *quantum logical gate*, is inspired by classical circuit and logic gates. Each quantum code has an equivalent quantum circuit; for example, the circuit in Figure 1b is equivalent to the code in Figure 1a applied for 4-qubits.

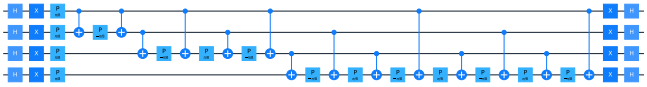
Despite the direct use of quantum bits and quantum gates in quantum programming, which gives the impression of low-level programming, the approach provides feature-rich high-level programming [22], producing operations that cannot be executed directly by a quantum computer and must undergo a compilation process. For example, in Figure 1b, note that the quantum gate in the middle of the circuit, a multi-controlled Pauli Z gate, encompasses all 4 qubits. This type of multi-qubit gate cannot be executed directly by a quantum computer and therefore must be decomposed into a sequence of one- and two-qubit gates with equivalent effect, as shown in Figure 1c. This is analogous to a line of high-level classical code being decomposed into several lines of assembly code.

```
def diffusion(q : Quant):
    with around(cat(H, X), q):
        with control(q[:-1]):
            Z(q[-1])
```

(a) Ket code.



(b) Quantum Circuit.



(c) Decomposed Circuit.

Figure 1: Grover diffusion operation implemented using Ket (a), along with its corresponding high-level (b) and decomposed (c) quantum circuit for 4 qubits.

The quantum compilation process can be split into three main parts: quantum gate decomposition [3, 8, 14, 15, 22, 23, 27, 32], where multi-qubit gates are broken down into a sequence of one- and two-qubit operations; circuit mapping [4, 16, 20, 28, 30, 31], where the logical qubits are mapped to physical qubits that are

limited in their connectivity, thereby restricting which qubits can participate in a two-qubit gate; and lastly, a pulse schedule is generated based on the mapped circuit [2, 13, 25]. These pulses are then passed to an arbitrary waveform generator (AWG) that physically controls the qubits. Each step of this decomposition process brings the quantum code closer to the hardware execution and becomes more hardware-dependent.

This paper addresses the first step of the quantum compiler, namely the quantum gate decomposition, by providing an analysis of the trade-offs between different quantum gate decomposition algorithms in terms of compilation time and execution time. Unlike classical compilation, where code is compiled once and executed many times, quantum compilation often occurs during runtime, making this trade-off particularly significant in quantum programming.

We implemented state-of-the-art quantum gate decomposition algorithms in Ket's quantum compiler, Libket, and evaluated the compilation and execution times of each algorithm. There is no definitive answer as to whether prioritizing execution time over compilation time, or *vice-versa*, is preferable; the best choice depends on the specifications of the classical and quantum hardware. The objective of this paper is to provide a basis for helping quantum compiler developers and programmers make informed decisions.

The main contributions of this paper are:

- A comprehensive survey of quantum gate decomposition algorithms, including optimizations beyond those proposed by the original authors.
- Implementation of all surveyed algorithms in the Ket quantum compiler–marking the first unified implementation on a single platform and enabling consistent benchmarking.
- A classification of the algorithms into two categories, forming the basis for two quantum compilation profiles: one focused on minimizing compilation time, the other on reducing quantum execution time.

The remainder of this paper is organized as follows. Section 2 introduces classical and quantum runtime, highlighting the implications for quantum program compilation. Section 3 examines high-level quantum programming and shows how multi-qubit gates naturally arise in such code. Section 4 then surveys the state-of-the-art decomposition algorithms used to break down these gates into executable instructions. Section 5 follows with a performance evaluation of these algorithms, focusing on CNOT count and circuit depth. Based on the benchmark data, Section 6 analyzes the trade-offs involved and introduces two quantum compilation profiles. Finally, Section 7 presents our conclusions and final remarks.

2 Classical and Quantum Runtime

A quantum program is not entirely quantum; rather, it is a classical-quantum program. Classical processing is always required at least to prepare the quantum circuit inputs and process the quantum execution outputs. The quantum computer can be seen as an accelerated processing unit, similar to an FPGA or GPU, where the CPU manages the execution. This implies that a quantum application has two distinct runtimes: a classical runtime, which includes all program execution, and a quantum runtime, which occurs within it, with a program potentially initiating several quantum runtimes.

Unlike classical code, which can be compiled once and reused with different inputs, a quantum circuit must be compiled separately for each specific input. This means that a quantum program designed to work with varying inputs can only be constructed at classical runtime–after all necessary parameters are known. As a result, the quantum portion of a classical-quantum program cannot be compiled in advance.

For instance, in Figure 1a, the function `diffusion` takes a list of qubits as input. In classical compilation, this would typically generate a loop whose size depends on the input list. However, for quantum compilation, the exact number of qubits must be known in advance to generate the correct quantum circuit. This requirement means the quantum compiler must be invoked dynamically at classical runtime–just before executing the quantum code.

The performance of a quantum application is evaluated by the number of two-qubit gates in the compiled quantum circuit. This two-qubit gate is typically a CNOT gate[1]. Single-qubit gates are generally not counted, as sequences of such gates can often be merged into a single equivalent operation. The time required to execute quantum code on a quantum computer is directly related to the circuit depth. Since gates acting on independent qubits can be executed in parallel, the execution time may be shorter than the total number of CNOT gates. For this reason, circuit depth serves as a metric for quantum execution time. Conversely, the number of CNOTs impacts compilation time, as each CNOT represents a computational step for the quantum compiler.

Since the compilation of the quantum circuit occurs at classical runtime, the trade-off between compilation time and quantum execution time directly impacts the overall classical-quantum execution time. This trade-off can also be viewed as a balance between classical computation (compilation) and quantum execution time.

3 High-Level Quantum Programming

While quantum programming introduces unique challenges–such as the impossibility of copying quantum data due to the no-cloning theorem [29]–it also offers powerful abstractions that simplify the development of quantum applications. One such feature is the ability to automatically invoke the inverse of a quantum operation. In this paper, we examine quantum programming through the lens of Ket[2], an open-source quantum programming platform with a Python API [7].

At its core, Ket provides a minimal yet expressive set of quantum gates: the Pauli gates (`X`, `Y`, and `Z`); Rotation gates (`RX`, `RY`, and `RZ`); the Phase gate (`P`); and the Hadamard gate (`H`). These gates are sufficient to prepare a qubit in any desired state. However, on their own, they cannot achieve universal quantum computation, as single-qubit gates cannot generate entanglement. To address this limitation, Ket allows any gate to be applied with one or more control qubits, enabling the creation of entangled states.

By adding just a controlled NOT[3] to the basic set of quantum gates, universal quantum computation becomes achievable. In Ket, the CNOT gate is provided by a function of the same name, but it

[1] All analyses in this paper are also valid if CZ gates are used instead.
[2] https://quantumket.org
[3] Also known as the controlled Pauli `X` or simply the CNOT gate.

can also be constructed using the function `ctrl` as

```
lambda c, t: ctrl(c, X)(t),
```

or by using the "`with control`" construction. Any quantum gate or function that contains quantum gates–as in the one shown in Figure 1a–can be called with control qubits. This flexibility to append control qubits to any quantum gate greatly enhances high-level quantum programming by enabling the easy construction of complex operations from basic quantum gates.

A controlled quantum gate call is the quantum analog of the classical `if` statement, where a gate is applied to the target qubits only if the control qubits are all in the state $|1\rangle$. The key difference is that a controlled operation acts on the superposition and has the ability to create entanglement.

The code in Figure 2 illustrates how controlled operations can appear in high-level quantum programming. The function `prepare` takes a list of qubits alongside a list of measurement probabilities $[r_k]$ and a list of phase values $[\theta_k]$, and prepares the qubits in the state $\sum_k \sqrt{r_k}\, e^{i\theta_k} |k\rangle$. This function calls itself recursively, adding a control qubit at each recursion. Therefore, even though there is no direct call for controlled `RY` or `P` gates, the execution of this code creates several controlled operations.

```python
def prepare(
    qubits: Quant,
    prob: ParamTree | list[float],
    amp: list[float] | None = None,
):
    if not isinstance(prob, ParamTree):
        prob = ParamTree(prob, amp)
    head, *tail = qubits
    RY(prob.value, head)
    if prob.is_leaf():
        with around(X, head):
            P(prob.phase0, head)
        return P(prob.phase1, head)
    with around(X, head):
        ctrl(head, prepare)(tail, prob.left)
    ctrl(head, prepare)(tail, prob.right)
```

Figure 2: Arbitrary quantum state preparation algorithm implemented using Ket. For the implementation of the `ParamTree`, see reference Rosa et al. [22, Figure 10a].

Alongside controlled gate calls, Ket provides constructions that take advantage of the reversibility of quantum computation. A key example is the "`with around`" construction, which acts as a basis change–analogous to basis changes in linear algebra–for the inner scope. In the code of Figure 2, the `with around` construction is used to change the control state from $|1\rangle$ to $|0\rangle$. This construction also allows the compiler to remove some controlled gate calls, for the code from Figure 2, those associated with the `X` gate [22].

4 Decomposition Algorithms

In Ket, multi-qubit gates are constructed such that only multi-controlled versions of the gates `X`, `Y`, `Z`, `RX`, `RY`, `RZ`, `P`, and `H` arise from high-level programming. This constraint allows the quantum compiler to implement a streamlined and efficient set of decomposition algorithms. We categorize these gates into three distinct groups based on their decomposition requirements: (i) Pauli Gates (`X`, `Y`, and `Z`), a decomposition algorithm for one of these gates can be applied to the others with a simple basis change on the target qubit; (ii) Rotation Gates (`RX`, `RY`, and `RZ`), these belong to the special unitary group SU(2); (iii) Phase and Hadamard Gates (`P` and `H`), these belong to the broader unitary group U(2)[4].

Table 1 summarizes state-of-the-art quantum gate decomposition algorithms. Some of these algorithms require *auxiliary qubits*, which are additional qubits beyond those directly involved in the controlled gate. Depending on the algorithm, auxiliary qubits must either be initialized in the $|0\rangle$ state (Clean auxiliary) or can be in any state (Dirty auxiliary). Regardless of their initial state, auxiliary qubits must be restored to their original state after decomposition so that they can be reused elsewhere and do not create new entanglements. Typically, decompositions using more auxiliary and/or clean qubits are more efficient. The Ket compiler automatically manages the allocation and usage of auxiliary qubits when available [23].

Table 1 also presents algorithm variations, such as C2X and C3X, which use approximations of 2-controlled and 3-controlled NOT gates [17], respectively. Additionally, some algorithms are labeled *Linear* or *Log*, which are algorithms that share similar auxiliary requirements.

The quantum circuit depth and the number of CNOTs presented in Table 1 were obtained by fitting the curves of the benchmark data. In cases where a good curve fit was not possible, we present the complexity as stated by the authors in big-O notation.

In the following subsections, we discuss the decomposition algorithms required for each class of quantum gates. In Section 5, we present a performance comparison of the decomposition algorithms.

4.1 Pauli Gates

The Pauli gates are among the most commonly used quantum gates in algorithm design. A key insight for gate decomposition is that the decomposition of one Pauli gate can be adapted for the others. For example, in Figure 4a, the `Y` and `Z` gates are defined using the `X` gate with a basis change. Consequently, the circuits for their controlled versions, shown in Figures 4b and 4c, rely on a controlled `X` gate. This example explicitly illustrates the behavior of Ket's compiler, which automatically applies this principle: only the decomposition of the Pauli `X` gate is explicitly defined, while the other Pauli gates are implemented through basis changes. In this paper, we evaluate only the performance of the `X` gate decomposition, as the decomposition of the other Pauli gates requires only an additional two single-qubit gates.

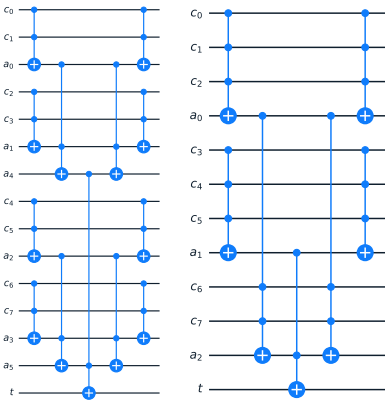
Network. The Network decomposition [19, p. 183] is one of the most efficient algorithms and can be applied to any multi-controlled gate. Figure 3 illustrates two variants of the algorithm for a 8-controlled Pauli `X` gate. Note that, in addition to the target and control qubits, this decomposition requires auxiliary qubits, all initialized to the $|0\rangle$ state. For Pauli gate decomposition, we can reduces the number of auxiliary qubits by one or two compared to other gates.

This decomposition results in a circuit depth $O(\log(n))$ for an n-controlled gate when organizing the gates as presented by Maslov

[4] All single-qubit gates are in U(2), with Rotation gates in SU(2) $\subset$ U(2).

Table 1: Gate decomposition algorithms evaluated in this paper. *No. Aux* denotes the number of auxiliary qubits required, and *Aux State* specifies their state. *Depth*, *No. CNOTs*, and *No. Aux* are reported for an n-controlled gate decomposition.

Gates	Algorithms	Mode	No. Aux	Aux State	Depth	No. CNOTs
Pauli Gates	V Chain	C2X	$n-2$	Clean	$4n$	$6n$
				Dirty	$8n$	$8n$
		C3X	$\lceil \frac{n-3}{2} \rceil$	Clean	$6n$	$6n$
				Dirty	$12n$	$12n$
	Single Aux	Linear	1	Clean/Dirty	$8n$	$12n$
		Log	1	Clean	$O(\log(n)^3)$	$O(n\log(n)^4)$
				Dirty	$O(\log(n)^3)$	$O(n\log(n)^4)$
Rotation Gates	SU(2)	Linear	0	N/A	$8n$	$12n$
		Log	0	N/A	$O(\log(n)^3)$	$O(n\log(n)^4)$
Phase and Hadamard	SU(2) Rewrite		1	Clean	$8n$	$12n$
	Single Aux U(2)		1	Clean	$O(\log(n)^3)$	$O(n\log(n)^4)$
U(2)	Network	C2X	$\approx n$	Clean	$6\log_2(n)$	$6n$
		C3X	$\approx \frac{n}{2}$	Clean	$8\log_2(n)$	$6n$
	Liner Depth		0	N/A	$16n$	$n^2/10$



(a) Network C2X. **(b) Network C3X.**

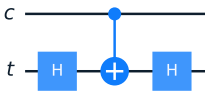
Figure 3: Network decomposition for 8-controlled Pauli X.

```
def myY(q):
    with around(SD, q):
        X(q)
def myZ(q):
    with around(H, q):
        X(q)

def myCY(c, t):
    ctrl(c, myY)(t)
def myCZ(c, t):
    ctrl(c, myZ)(t)
```

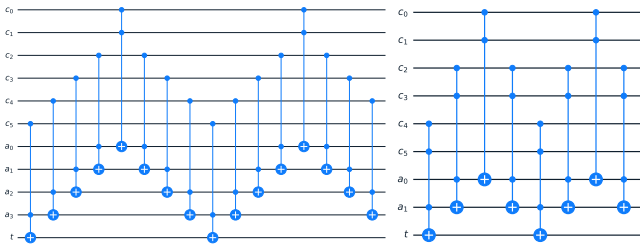
(a) Ket code.

(b) myCY Circuit.



(c) myCZ Circuit.

Figure 4: Defining the gates Y and Z using the X gate and a basis change, allowing the controlled X gate to implement the controlled versions of these gates.



(a) V Chain C2X, Dirty. **(b) V Chain C3X, Dirty.**

Figure 5: V Chain decomposition for 6-controlled Pauli X.

[17, Corollary 5]. Additionally, for the decomposition of 2- and 3-controlled X gates, where the original target qubit is not directly affected, approximate versions of this decomposition are applied [17].

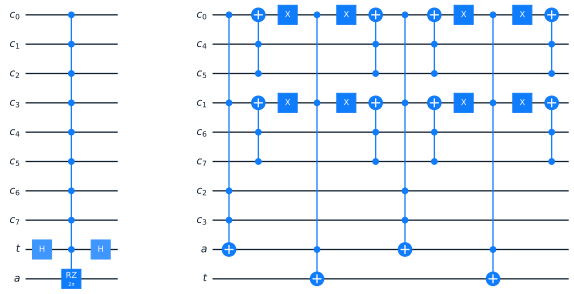
V Chain. The V Chain decomposition [3] has a structure similar to the Network decomposition for Pauli gates but allows the use of either dirty or clean auxiliary qubits. Like the Network decomposition, the V Chain decomposition utilizes approximate 2- or 3-controlled X. Although this algorithm results in a number of CNOT gates similar to the Network decomposition, especially when using clean auxiliary qubits, it produces a circuit with linear depth rather than logarithmic depth. Figure 5 illustrates a V Chain decomposition using dirty auxiliary qubits for a 6-controlled X gate.

Single Aux. The Single Aux decomposition algorithms require only a single auxiliary qubit and include two methods: one that results in a quantum circuit with linear depth, proposed by Zindorf and Bose [32], and another that achieves logarithmic depth, proposed by Claudon et al. [5].

The *Linear* algorithm relies on the application of a multi-controlled 2π rotation gate into an auxiliary qubit, as illustrated in Figure 6a. This results in a multi-controlled Z gate with target and control qubits, which can be transformed into a multi-controlled X via a basis change at the target. The decomposition of the multi-controlled rotation is performed without auxiliary with the SU(2) Linear algorithm [32].

The Single Aux Log decomposition [5] recursively breaks an n-controlled X gate into $2\sqrt{n}$-controlled X gates that can execute in parallel. The base case of the recursion is a 4-controlled X gate, which can be decomposed without auxiliary qubits. Unlike the Linear algorithm, the number of CNOTs and the circuit depth can be reduced when using a clean auxiliary qubit. Figure 6b illustrates the Single Aux Log decomposition of an 8-controlled X gate.

Linear Depth. The Linear Depth decomposition algorithm proposed by Da Silva and Park [8], as the name suggests, results in a circuit with linear depth. This is an efficient algorithm that requires no auxiliary qubits and can be used for any quantum gate. We consider this algorithm as the fallback decomposition since it has no requirements to be applied to any U(2) gate. However, this algorithm results in a circuit with a quadratic number of CNOTs.



(a) Linear, Dirty. (b) Log, Dirty.

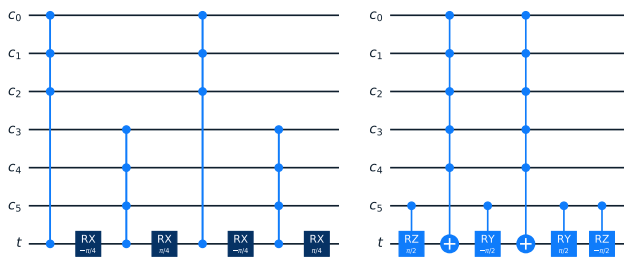
Figure 6: Single Aux decomposition for 8-controlled Pauli X.

4.2 Rotation Gates

Alongside the Network decomposition, Rotation gates allow for the use of the SU(2) decomposition algorithms, which requires no auxiliary qubits. We categorize these algorithms as SU(2) Linear, proposed by Zindorf and Bose [32], which is also used for the Single Aux Linear and SU(2) Rewrite decompositions [22, 32]; and SU(2) Log, which relies on the Single Aux Log decomposition algorithm for Pauli gates [5].

The SU(2) Linear decomposition algorithm uses an approximate version of a multi-controlled Z gate, as illustrated in Figure 7a. Each multi-controlled Z gate is decomposed into an approximate version where free qubits serve as auxiliaries. The approximate decomposition of the Z gate relies on phase differences canceling out in pairs.

Figure 7b illustrates the decomposition using the SU(2) Log algorithm, where each multi-controlled X is decomposed using the Single Aux Log algorithm with a dirty auxiliary, and the single-controlled gates can be decomposed without an auxiliary qubit using just two CNOTs. Figure 7 illustrates the decomposition of a multi-controlled RX(π) gate. For other Rotation gates and angles, the multi-controlled gates are arranged in the same structure.



(a) SU(2) Linear. (b) SU(2) Log.

Figure 7: SU(2) decomposition for 6-controlled RX(π).

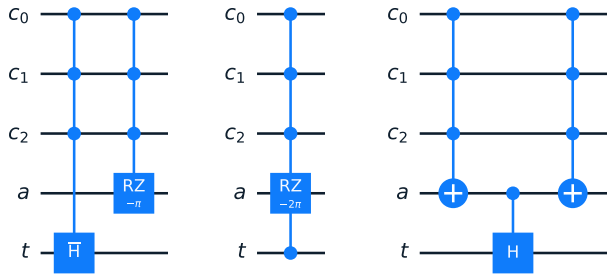
4.3 Phase and Hadamard Gates

When multiple auxiliary qubits, or none, are available, the Network and Linear Depth decompositions can be used, respectively, as for most gates. However, when only a single auxiliary qubit is available, there are two decomposition algorithms available: SU(2) Rewrite [22], which leverages the SU(2) Linear decomposition [32], and the Single Aux U(2), which uses the Single Aux Log decomposition [5].

The SU(2) Rewrite decomposition relies on the fact that any U(2) gate can be represented by a SU(2) gate and a global phase, and that this global phase can be "corrected" by a multi-controlled RZ gate [22]. Figure 8a illustrates the decomposition of a multi-controlled Hadamard gate. For this decomposition, the two multi-controlled gates can be fused, resulting in the same complexity as decomposing a single multi-controlled SU(2) gate [32]. For the multi-controlled Phase gate, illustrated in Figure 8b, the decomposition can be simplified, but this results only in a constant reduction in the number of CNOTs [32].

For the Single Aux U(2) decomposition, a clean auxiliary qubit is added to break the multi-controlled gate into two multi-controlled X gates and a dirty auxiliary, as illustrated in Figure 8c. This decomposition results in a circuit of logarithmic depth since the multi-controlled X gates are then decomposed using the Single Aux Log decomposition. The decomposition method applied to the Phase and Hadamard gates can be extended to any other gate, but in many cases, a more efficient decomposition is available for Pauli and Rotation gates.

The decompositions listed for the U(2) gate in Table 1 are used for any gate as presented in the previous sections, with the exception of the Linear Depth decomposition for Rotation gates. In the next section, we present a benchmark of the decomposition algorithms discussed in this section.



(a) SU(2) Rewrite. (b) SU(2) Rewrite. (c) Single Aux U(2).

Figure 8: Decomposition for a 3-controlled Hadamard (a and c) and a 3-controlled Phase(π) (b). In a, $\overline{\text{H}}$ is a SU(2) gate equivalent to H [22].

5 Algorithms Benchmark

We implemented all the decomposition algorithms presented in Table 1 in the Ket quantum programming platform. This is the first implementation that consolidates the current state-of-the-art decomposition algorithms into a single platform, allowing us to collect numerical data from their execution. Our objective is to classify the performance of these algorithms in terms of compilation time and quantum execution time, creating compilation profiles where those key metrics will be taken into consideration. We classify the algorithms into two profiles: *Compilation Time* and *Execution Time*.

The *Compilation Time* profile, as the name suggests, is optimized to reduce the classical execution time of the decomposition algorithms, which may also result in improved execution time for simulated quantum executions. The *Execution Time* profile targets large-scale Fault-Tolerant quantum computers [9, 11] with the potential to compute with millions of qubits.

As Ket's compiler automatically manages auxiliary qubits whenever they are available on the quantum computer [23], the data was gathered using high-level programming instructions such as "`ctrl(c, gate)(t)`", where `c` is a list of control qubits, `gate` is a single-qubit gate, and `t` is the target qubit. These qubits belong to a quantum processor with enough additional qubits available for use as auxiliary in the decomposition. No actual quantum execution was performed during the tests; only the decomposition was evaluated.

The primary metrics we evaluated in our benchmarks are the compilation and quantum execution times required to decompose an n-controlled quantum gate. We measure the compilation time based on the number of CNOTs and the quantum execution time based on the quantum circuit depth. The decomposition of multi-qubit gates, which is the focus of this study, is just one step in compiling a quantum program. Therefore, measuring the time in seconds to compile the code or decompose the gates may not be a suitable metric for compilation time. Instead, the number of CNOTs allows us to infer the impact of the decomposition on subsequent compilation steps, which have time complexity directly related to the number of CNOTs.

We evaluate the quantum execution time in terms of quantum circuit depth. Since quantum gates that do not depend on each other can execute in parallel, the depth represents the minimum execution time required for the circuit. For the circuit depth metric, we consider only CNOT gates, as they have the most significant impact on quantum execution time. We disregard single-qubit gates, since sequences of such gates can be fused into a single operation, and in some cases, they can be virtually implemented [18] with no impact on execution time. Also, the actual quantum execution time, in seconds, depends on hardware constraints such as qubit connectivity, which also affects the compilation step related to circuit mapping, as well as the availability of hardware resources to perform operations in parallel.

We organize our results using the same gate categories introduced in the previous section: Pauli gates, Rotation gates, and Phase and Hadamard gates. Next, we present the benchmark results for each group, followed by our analysis in Section 6.

Pauli Gates Benchmark. Figure 9 presents the number of CNOTs and circuit depth for multi-controlled Pauli gates. The data was obtained with the instruction "`ctrl(c, X)(t)`", but for the other Pauli gates, there is a difference of only a constant number of single-qubit gates, which does not affect the presented data.

Rotation Gates Benchmark. Figure 10 presents the number of CNOTs and circuit depth for multi-controlled Rotation gates. The data was obtained using the instruction "`ctrl(c, RX(pi/2))(t)`", but for the other Rotation gates, there is a difference of only a constant number of single-qubit gates, which does not affect the presented data.

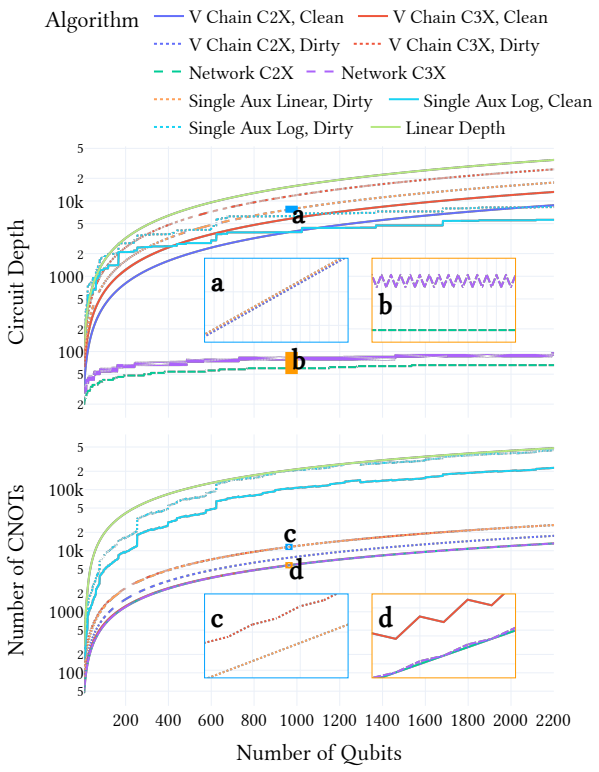


Figure 9: Comparison of various decomposition algorithms for *Multi-Controlled Pauli Gates* across two metrics: Circuit Depth and Number of CNOTs. Insets display zoomed-in views for specific regions with matching colors.

Phase and Hadamard Gates Benchmark. Figure 11 presents the number of CNOTs and circuit depth for multi-controlled Phase and Hadamard gates. The data was obtained with the instruction "`ctrl(c, H)(t)`", but for the Phase gates, there is a difference of only a constant number of single-qubit gates, which does not significantly affect the data.

6 Results Analysis

Based on the data presented in the previous section, Table 2 presents the quantum gate decomposition algorithms from Table 1, classified into the two compilation profiles. Note that not all algorithms are included in Table 2, as some decompositions with fewer auxiliary qubit requirements may be more efficient.

The benchmark results reveal that the Network decomposition stands out as the most efficient algorithm across all gate types. However, this performance comes at the cost of requiring the largest number of clean auxiliary qubits. Interestingly, the initial assumption that increasing the number of auxiliary qubits would always lead to more efficient decompositions does not consistently hold. In some cases, additional auxiliaries provide no performance improvements.

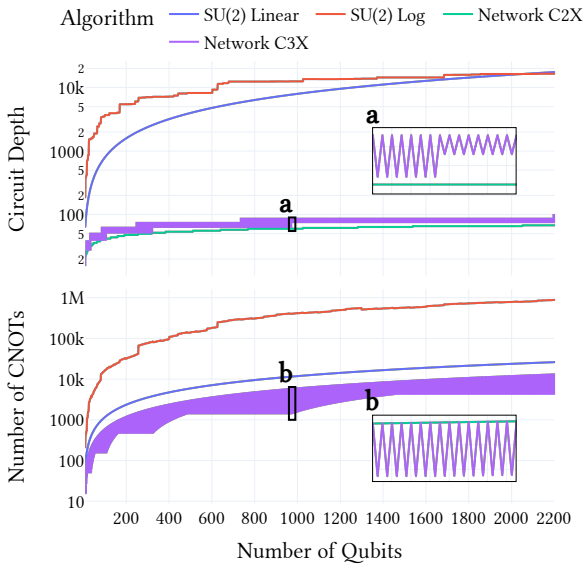


Figure 10: Comparison of various decomposition algorithms for *Multi-controlled Rotation Gates* across two metrics: Circuit Depth and Number of CNOTs. Insets display zoomed-in views for specific regions.

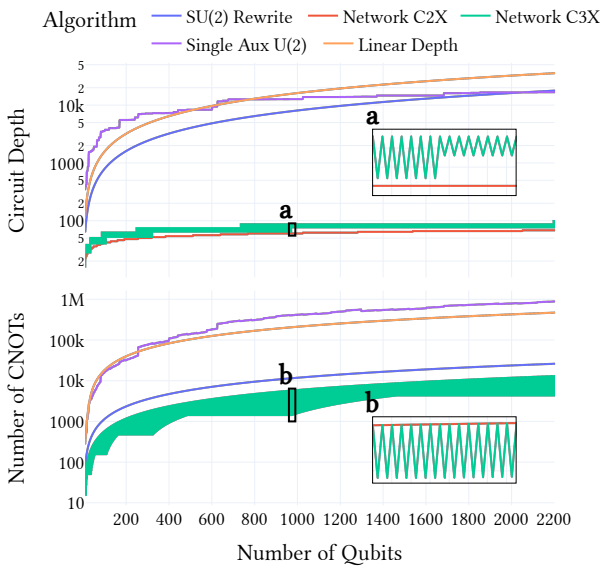


Figure 11: Comparison across different decomposition algorithms for *Multi-controlled Phase and Hadamard Gates* across two metrics: Circuit Depth and Number of CNOTs. Insets display zoomed-in views for specific regions.

Table 2: Quantum gate decomposition algorithms ranked by priority for two compilation profiles.

Gate	Compilation Profile	
	Compilation Time	Execution Time
Pauli Gates	1st Network C2X 2nd Network C3X 3rd V Chain C3X, Clean 4th V Chain C2X, Dirty 5th Single Aux Linear 6th Linear Depth	1st Network C2X 2nd Network C3X 3rd Single Aux Log, Clean 4th Single Aux Log, Dirty 5th Linear Depth
Rotation Gates	1st Network C3X 2nd SU(2) Linear	1st Network C2X 2nd Network C3X 3rd SU(2) Log
Phase and Hadamard	1st Network C3X 2nd SU(2) Rewrite 3rd Linear Depth	1st Network C2X 2nd Network C3X 3rd Single Aux U(2) 4th Linear Depth

Algorithms that produce logarithmic circuit depth demonstrate their advantages only in scenarios involving more than 1000 qubits. Despite their depth efficiency, these algorithms tend to perform poorly in terms of CNOT gate count, often ranking alongside the Linear Depth decomposition as the least efficient in this regard.

With the exception of Rotation gates–which can be decomposed without auxiliary qubits and still maintain a linear number of CNOTs–most other gate types fall back on the Linear Depth decomposition when no auxiliary qubits are available. This makes Linear Depth a crucial strategy for current quantum compilers. However, as future quantum devices provide access to more qubits, the relevance of this method may decline.

A particularly noteworthy result from the benchmarks is the significant performance gap between using clean versus dirty auxiliary qubits. The data suggest that having access to a quantum processor with twice the number of qubits as needed by the program can dramatically reduce both compilation time and quantum execution time. The Single Aux Linear algorithm is an exception, as its performance remains unaffected by the auxiliary qubit state.

7 Final Remarks

In this paper, we addressed the trade-off between compilation time and quantum execution time in quantum programs. We argue that this trade-off is more important for quantum applications than for classical ones, as the compilation of a quantum program cannot be performed *a priori*.

Our analysis focuses on quantum gate decomposition, which is the first step in quantum code compilation. We have gathered data from the state-of-the-art quantum gate decomposition algorithms, which we implemented on the Ket quantum programming platform. To the best of our knowledge, this is the first implementation of all these algorithms within a single platform.

From the data presented in Section 5, we created two compilation profiles: one focused on reducing compilation time and the other on reducing quantum execution time. These results are presented in Table 2. The quantum execution time profile considers large-scale quantum computers that go beyond current capabilities but will be necessary to run algorithms such as Shor's algorithm [24],

which could break RSA encryption [10]. Additionally, the presented data can be used to construct a lookup table that selects the best decomposition algorithm based on the number of qubits involved in a multi-qubit gate, as the performance may not be easily predictable for operations with fewer than 2200 qubits.

The results of this paper can help quantum compiler developers select the best decomposition algorithms for their compilers, depending on the type of gate and the number of control qubits. Note that the selection of the decomposition algorithm can be automatically performed by the compiler based on the available auxiliary qubits [23]. However, the findings from this paper can also aid quantum developers in identifying the performance associated with each multi-controlled gate, enabling them to make more informed decisions when selecting instructions to use.

In our study, we only evaluated the decomposition step of quantum compilation. However, circuit mapping may have a significant impact on the final performance of the quantum program. Circuit mapping adjusts the quantum program to the connectivity constraints of the quantum computer. This means that, regardless of the circuit mapping algorithm used, different quantum computer architectures (or qubit connectivity structures) may be better suited to different decomposition algorithms. Future work could analyze these decomposition algorithms across various qubit connectivity structures, such as line, grid, and torus configurations.

Another direction for future work is to analyze the performance of the decomposition algorithms in the context of non-Clifford gates [1], as these gates have a significant impact on quantum execution time when the program is encoded in a Quantum Error Correction code [21]. Additionally, examining the impact of circuit optimization techniques, such as ZX-calculus [6], on the final circuit would be an interesting direction for further research.

ARTIFACT AVAILABILITY

The authors declare that the research artifacts supporting the findings of this study are accessible at https://doi.org/10.5281/zenodo.16964846.

ACKNOWLEDGMENTS

ECRR acknowledges the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES, Finance Code 001; EID, JM, and ECRR acknowledges the Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq through grant number 409673/2022-6; JM, EID, and ECRR acknowledges the Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina - FAPESC through Project FAPESC TR nº 2024TR002672.

REFERENCES

[1] Scott Aaronson and Daniel Gottesman. 2004. Improved Simulation of Stabilizer Circuits. *Physical Review A* 70, 5 (Nov. 2004), 052328. doi:10.1103/PhysRevA.70.052328

[2] Thomas Alexander, Naoki Kanazawa, Daniel J Egger, Lauren Capelluto, Christopher J Wood, Ali Javadi-Abhari, and David C McKay. 2020. Qiskit Pulse: Programming Quantum Computers through the Cloud with Pulses. *Quantum Science and Technology* 5, 4 (Aug. 2020), 044006. doi:10.1088/2058-9565/aba404

[3] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. 1995. Elementary Gates for Quantum Computation. *Physical Review A* 52, 5 (Nov. 1995), 3457–3467. doi:10.1103/PhysRevA.52.3457

[4] Sohini Chowdhury, Rupali Gill, Arti Badhoutiya, Arun Pratap Srivastava, Akhilesh Kumar Khan, and Rajesh Singh. 2024. Qubit Allocation Strategies in Quantum Computing for Improved Computational Efficiency. In *2024 4th International Conference on Innovative Practices in Technology and Management (ICIPTM)*. IEEE, Noida, India, 1–6. doi:10.1109/ICIPTM59628.2024.10563524

[5] Baptiste Claudon, Julien Zylberman, César Feniou, Fabrice Debbasch, Alberto Peruzzo, and Jean-Philip Piquemal. 2024. Polylogarithmic-Depth Controlled-NOT Gates without Ancilla Qubits. *Nature Communications* 15, 1 (July 2024), 5886. doi:10.1038/s41467-024-50065-x

[6] Bob Coecke and Ross Duncan. 2011. Interacting Quantum Observables: Categorical Algebra and Diagrammatics. *New Journal of Physics* 13, 4 (April 2011), 043016. doi:10.1088/1367-2630/13/4/043016

[7] Evandro Chagas Ribeiro Da Rosa and Rafael De Santiago. 2022. Ket Quantum Programming. *ACM Journal on Emerging Technologies in Computing Systems* 18, 1 (Jan. 2022), 1–25. doi:10.1145/3474224

[8] Adenilton J. Da Silva and Daniel K. Park. 2022. Linear-Depth Quantum Circuits for Multiqubit Controlled Gates. *Physical Review A* 106, 4 (Oct. 2022), 042602. doi:10.1103/PhysRevA.106.042602

[9] Simon J Devitt, William J Munro, and Kae Nemoto. 2013. Quantum Error Correction for Beginners. *Reports on Progress in Physics* 76, 7 (July 2013), 076001. doi:10.1088/0034-4885/76/7/076001

[10] Craig Gidney and Martin Ekerå. 2021. How to Factor 2048 Bit RSA Integers in 8 Hours Using 20 Million Noisy Qubits. *Quantum* 5 (April 2021), 433. doi:10.22331/q-2021-04-15-433

[11] Google Quantum AI and Collaborators, Rajeev Acharya, Dmitry A. Abanin, Laleh Aghababaie-Beni, Igor Aleiner, Trond I. Andersen, Markus Ansmann, Frank Arute, Kunal Arya, Abraham Asfaw, Nikita Astrakhantsev, Juan Atalaya, Ryan Babbush, Dave Bacon, Brian Ballard, Joseph C. Bardin, Johannes Bausch, Andreas Bengtsson, Alexander Bilmes, Sam Blackwell, Sergio Boixo, Gina Bortoli, Alexandre Bourassa, Jenna Bovaird, Leon Brill, Michael Broughton, David A. Browne, Brett Buchea, Bob B. Buckley, David A. Buell, Tim Burger, Brian Burkett, Nicholas Bushnell, Anthony Cabrera, Juan Campero, Hung-Shen Chang, Yu Chen, Zijun Chen, Ben Chiaro, Desmond Chik, Charina Chou, Jahan Claes, Agnetta Y. Cleland, Josh Cogan, Roberto Collins, Paul Conner, William Courtney, Alexander L. Crook, Ben Curtin, Sayan Das, Alex Davies, Laura De Lorenzo, Dripto M. Debroy, Sean Demura, Michel Devoret, Agustin Di Paolo, Paul Donohoe, Ilya Drozdov, Andrew Dunsworth, Clint Earle, Thomas Edlich, Alec Eickbusch, Aviv Moshe Elbag, Mahmoud Elzouka, Catherine Erickson, Lara Faoro, Edward Farhi, Vinicius S. Ferreira, Leslie Flores Burgos, Ebrahim Forati, Austin G. Fowler, Brooks Foxen, Suhas Ganjam, Gonzalo Garcia, Robert Gasca, Elie Genois, William Giang, Craig Gidney, Dar Gilboa, Raja Gosula, Alejandro Grajales Dau, Dietrich Graumann, Alex Greene, Jonathan A. Gross, Steve Habegger, John Hall, Michael C. Hamilton, Monica Hansen, Matthew P. Harrigan, Sean D. Harrington, Francisco J. H. Heras, Stephen Heslin, Paula Heu, Oscar Higgott, Gordon Hill, Jeremy Hilton, George Holland, Sabrina Hong, Hsin-Yuan Huang, Ashley Huff, William J. Huggins, Lev B. Ioffe, Sergei V. Isakov, Justin Iveland, Evan Jeffrey, Zhang Jiang, Cody Jones, Stephen Jordan, Chaitali Joshi, Pavol Juhas, Dvir Kafri, Hui Kang, Amir H. Karamlou, Kostyantyn Kechedzhi, Julian Kelly, Trupti Khaire, Tanuj Khattar, Mostafa Khezri, Seon Kim, Paul V. Klimov, Andrey R. Klots, Bryce Kobrin, Pushmeet Kohli, Alexander N. Korotkov, Fedor Kostritsa, Robin Kothari, Borislav Kozlovskii, John Mark Kreikebaum, Vladislav D. Kurilovich, Nathan Lacroix, David Landhuis, Tiano Lange-Dei, Brandon W. Langley, Pavel Laptev, Kim-Ming Lau, Loïck Le Guevel, Justin Ledford, Joonho Lee, Kenny Lee, Yuri D. Lensky, Shannon Leon, Brian J. Lester, Wing Yan Li, Yin Li, Alexander T. Lill, Wayne Liu, William P. Livingston, Aditya Locharla, Erik Lucero, Daniel Lundahl, Aaron Lunt, Sid Madhuk, Fionn D. Malone, Ashley Maloney, Salvatore Mandrà, James Manyika, Leigh S. Martin, Orion Martin, Steven Martin, Cameron Maxfield, Jarrod R. McClean, Matt McEwen, Seneca Meeks, Anthony Megrant, Xiao Mi, Kevin C. Miao, Amanda Mieszala, Reza Molavi, Sebastian Molina, Shirin Montazeri, Alexis Morvan, Ramis Movassagh, Wojciech Mruczkiewicz, Ofer Naaman, Matthew Neeley, Charles Neill, Ani Nersisyan, Hartmut Neven, Michael Newman, Jiun How Ng, Anthony Nguyen, Murray Nguyen, Chia-Hung Ni, Murphy Yuezhen Niu, Thomas E. O'Brien, William D. Oliver, Alex Opremcak, Kristoffer Ottosson, Andre Petukhov, Alex Pizzuto, John Platt, Rebecca Potter, Orion Pritchard, Leonid P. Pryadko, Chris Quintana, Ganesh Ramachandran, Matthew J. Reagor, John Redding, David M. Rhodes, Gabrielle Roberts, Eliott Rosenberg, Emma Rosenfeld, Pedram Roushan, Nicholas C. Rubin, Negar Saei, Daniel Sank, Kannan Sankaragomathi, Kevin J. Satzinger, Henry F. Schurkus, Christopher Schuster, Andrew W. Senior, Michael J. Shearn, Aaron Shorter, Noah Shutty, Vladimir Shvarts, Shraddha Singh, Volodymyr Sivak, Jindra Skruzny, Spencer Small, Vadim Smelyanskiy, W. Clarke Smith, Rolando D. Somma, Sofia Springer, George Sterling, Doug Strain, Jordan Suchard, Aaron Szasz, Alex Sztein, Douglas Thor, Alfredo Torres, M. Mert Torunbalci, Abeer Vaishnav, Justin Vargas, Sergey Vdovichev, Guifre Vidal, Benjamin Villalonga, Catherine Vollgraff Heidweiller, Steven Waltman, Shannon X. Wang, Brayden Ware, Kate Weber, Travis Weidel, Theodore White, Kristi Wong, Bryan W. K. Woo, Cheng Xing, Z. Jamie Yao, Ping Yeh, Bicheng Ying, Juhwan Yoo, Noureldin Yosri, Grayson Young, Adam Zalcman, Yaxing Zhang, Ningfeng Zhu, and Nicholas Zobrist. 2025. Quantum Error Correction below the Surface Code Threshold. *Nature* 638, 8052 (Feb. 2025), 920–926. doi:10.1038/s41586-024-08449-y

[12] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing - STOC '96*. ACM Press, Philadelphia, Pennsylvania, United States, 212–219. doi:10.1145/237814.237866

[13] You Huang, Mohammad T Amawi, Francesco Poggiali, Fazhan Shi, Jiangfeng Du, and Friedemann Reinhard. 2023. Calibrating Single-Qubit Gates by a Two-Dimensional Rabi Oscillation. *AIP Advances* 13, 3 (March 2023), 035226. doi:10.1063/5.0139454

[14] Raban Iten, Roger Colbeck, Ivan Kukuljan, Jonathan Home, and Matthias Christandl. 2016. Quantum Circuits for Isometries. *Physical Review A* 93, 3 (March 2016), 032318. doi:10.1103/PhysRevA.93.032318

[15] Raban Iten, Oliver Reardon-Smith, Emanuel Malvetti, Luca Mondada, Gabrielle Pauvert, Ethan Redmond, Ravjot Singh Kohli, and Roger Colbeck. 2021. Introduction to UniversalQCompiler. arXiv:1904.01072 [quant-ph] doi:10.48550/arXiv.1904.01072

[16] Gushu Li, Yufei Ding, and Yuan Xie. 2019. Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, Providence RI USA, 1001–1014. doi:10.1145/3297858.3304023

[17] Dmitri Maslov. 2016. Advantages of Using Relative-Phase Toffoli Gates with an Application to Multiple Control Toffoli Optimization. *Physical Review A* 93, 2 (Feb. 2016), 022311. doi:10.1103/PhysRevA.93.022311

[18] David C. McKay, Christopher J. Wood, Sarah Sheldon, Jerry M. Chow, and Jay M. Gambetta. 2017. Efficient Z Gates for Quantum Computing. *Physical Review A* 96, 2 (Aug. 2017), 022330. doi:10.1103/PhysRevA.96.022330

[19] Michael A. Nielsen and Isaac L. Chuang. 2010. *Quantum Computation and Quantum Information* (10th anniversary edition ed.). Cambridge university press, Cambridge. doi:10.1017/CBO9780511976667

[20] Siyuan Niu, Adrien Suau, Gabriel Staffelbach, and Aida Todri-Sanial. 2020. A Hardware-Aware Heuristic for the Qubit Mapping Problem in the NISQ Era. *IEEE Transactions on Quantum Engineering* 1 (2020), 1–14. doi:10.1109/TQE.2020.3026544

[21] Christophe Piveteau, David Sutter, Sergey Bravyi, Jay M. Gambetta, and Kristan Temme. 2021. Error Mitigation for Universal Gates on Encoded Qubits. *Physical Review Letters* 127, 20 (Nov. 2021), 200505. doi:10.1103/PhysRevLett.127.200505

[22] Evandro C. R. Rosa, Eduardo I. Duzzioni, and Rafael De Santiago. 2025. Optimizing Gate Decomposition for High-Level Quantum Programming. *Quantum* 9 (March 2025), 1659. doi:10.22331/q-2025-03-12-1659

[23] Evandro C. R. Rosa, Jerusa Marchi, Eduardo I. Duzzioni, and Rafael de Santiago. 2024. Automated Auxiliary Qubit Allocation in High-Level Quantum Programming. arXiv:2412.20543 [quant-ph] doi:10.48550/arXiv.2412.20543

[24] Peter W. Shor. 1997. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* 26, 5 (Oct. 1997), 1484–1509. doi:10.1137/S0097539795293172

[25] Leandro Stefanazzi, Kenneth Treptow, Neal Wilcer, Chris Stoughton, Collin Bradford, Sho Uemura, Silvia Zorzetti, Salvatore Montella, Gustavo Cancelo, Sara Sussman, Andrew Houck, Shefali Saxena, Horacio Arnaldi, Ankur Agrawal, Helin Zhang, Chunyang Ding, and David I. Schuster. 2022. The QICK (Quantum Instrumentation Control Kit): Readout and Control for Qubits and Detectors. *Review of Scientific Instruments* 93, 4 (April 2022), 044709. doi:10.1063/5.0076249

[26] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. 2018. Q#: Enabling Scalable Quantum Computing and Development with a High-level DSL. In *Proceedings of the Real World Domain Specific Languages Workshop 2018*. ACM, Vienna Austria, 1–10. doi:10.1145/3183895.3183901

[27] Rafaella Vale, Thiago Melo D. Azevedo, Ismael C. S. Araújo, Israel F. Araujo, and Adenilton J. Da Silva. 2024. Circuit Decomposition of Multicontrolled Special Unitary Single-Qubit Gates. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 3 (March 2024), 802–811. doi:10.1109/TCAD.2023.3327102

[28] Robert Wille and Lukas Burgholzer. 2023. MQT QMAP: Efficient Quantum Circuit Mapping. In *Proceedings of the 2023 International Symposium on Physical Design*. ACM, Virtual Event USA, 198–204. doi:10.1145/3569052.3578928

[29] W. K. Wootters and W. H. Zurek. 1982. A Single Quantum Cannot Be Cloned. *Nature* 299, 5886 (Oct. 1982), 802–803. doi:10.1038/299802a0

[30] Pengcheng Zhu, Weiping Ding, Lihua Wei, Xueyun Cheng, Zhijin Guan, and Shiguang Feng. 2023. A Variation-Aware Quantum Circuit Mapping Approach Based on Multi-Agent Cooperation. *IEEE Trans. Comput.* 72, 8 (Aug. 2023), 2237–2249. doi:10.1109/TC.2023.3242208

[31] Pengcheng Zhu, Zhijin Guan, and Xueyun Cheng. 2020. A Dynamic Look-Ahead Heuristic for the Qubit Mapping Problem of NISQ Computers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 12 (Dec. 2020), 4721–4735. doi:10.1109/TCAD.2020.2970594

[32] Ben Zindorf and Sougato Bose. 2024. Efficient Implementation of Multi-Controlled Quantum Gates. arXiv:2404.02279 [quant-ph] doi:10.48550/arXiv.2404.02279