Development of a multi-technology, template-based quantum circuits compilation toolchain

Quantum Information Processing (2022) 21:379
https://doi.org/10.1007/s11128-022-03649-9

Development of a multi-technology, template-based quantum circuits compilation toolchain

Manfredi Avitabile[1] · Giovanni Amedeo Cirillo[1] · Mario Simoni[1] · Giovanna Turvani[1] · Mariagrazia Graziano[2]

Received: 28 October 2021 / Accepted: 2 August 2022

Abstract
With noisy intermediate scale quantum computers (NISQ) becoming larger in scale and more reliable, quantum circuits are growing in size and complexity. In order to face the challenge of achieving optimal circuits, design automation approaches for improving and mapping quantum circuits on different architectures have been proposed, each one characterized by a specific optimization strategy. In this article, the use of a template-based approach for quantum circuits optimization purposes is explored, and the proposal of a modular compilation toolchain, which supports three quantum technologies (nuclear magnetic resonance, trapped ions and superconducting qubits), is presented. The toolchain tackles the task of implementing logic synthesis for single-qubit and multi-qubit gates in the compilation process and it is structured with multiple steps and modular libraries. The toolchain was tested through a benchmarking procedure, and the results for a subset of complex quantum circuits as inputs are here reported, alongside a comparison with those provided by the compilers of IBM's Qiskit and Cambridge Quantum Computing's t|ket⟩. The current toolchain prototype was crafted to be an easily expandable and reliable core for future developments,

✉ Giovanni Amedeo Cirillo
giovanni_cirillo@polito.it

✉ Giovanna Turvani
giovanna.turvani@polito.it

Manfredi Avitabile
manfredi.avitabile@studenti.polito.it

Mario Simoni
mario.simoni@polito.it

Mariagrazia Graziano
mariagrazia.graziano@polito.it

[1] Department of Electronics and Telecommunications, Politecnico di Torino, Corso Castelfidardo, 39, Torino 10129, Italy

[2] Department of Applied Science and Technology, Politecnico di Torino, Corso Duca degli Abruzzi, 24, Torino 10129, Italy

Published online: 12 November 2022

which could lead it to support even more quantum technologies and a fully fledged layout synthesis. Nonetheless, the obtained results are quite encouraging, and they prove that in certain conditions the Toolchain can be competitive in quantum circuits optimization, especially when dealing with single-qubit gates.

Keywords Noisy intermediate scale quantum (NISQ) · Quantum computing · Quantum circuits · Compilation toolchain · Template-based approach

1 Introduction

In an application scenario where quantum computing is going to be employed for facing concrete problems, in domains as Machine Learning, optimization, image processing or chemical simulations, manual quantum circuit design has become unpractical. To address the problem, the industry took inspiration from the **design automation** tools, vastly employed for the synthesis of digital circuits in classical computing, to produce refined, reliable quantum circuits to be adapted on the target devices. The goal is producing an optimized quantum circuit, equivalent to that of a certain quantum algorithm. The entity of the applied improvements and their focus may vary, depending on the desired performance, the target device and other degrees of freedom. These optimizations are usually performed by **compilers** [1], which take as input the high-level, abstract description of a certain application of a quantum algorithm or of a quantum circuit and automatically "translate" and adapt it, according to the given specifications. The resulting output is a quantum circuit built to work smoothly with the target device, tailored to properly optimize specific performance parameters.

Different figures of merit have been proposed for evaluating quantum circuit design and compilation. Among these, it is important to remind:

- The **circuit depth** [2, 3], which is the length of the longest sequence of quantum gates from qubit initialization to measurement. It can be seen as a quantum equivalent of the critical path of digital circuits.
- The **total gate count**, which is given by the total number of quantum gates belonging to a circuit. In some contexts, the gate count is done by distinguishing between single and two-qubit (usually CX) gates [4] or considering some application-specific critical gates, such as T and $T^\dagger$ in the context of fault-tolerant quantum computation [5, 6].

The quantum circuit shown in Fig. 1 has a circuit depth equal to 5 (or 6, depending if measurement is counted or not), associated with all the gates involving the lowest and the intermediate qubits, while the total gate count is equal to 5—when no distinction between single and two-qubit gates is done—otherwise is equal to the pair (2, 3), where the two values are equal to the occurrences of single and two-qubit gates, respectively. These metrics permit to estimate the complexity of a quantum circuit in a very simple way, but in some circumstances they could be quite limiting, since they do not assign different weights to different quantum gates, as it would be expected by considering the features of current quantum computing hardware. For example, the evaluation of the effects of *quantum gates duration* on the **circuit latency**, i.e., the total time required for

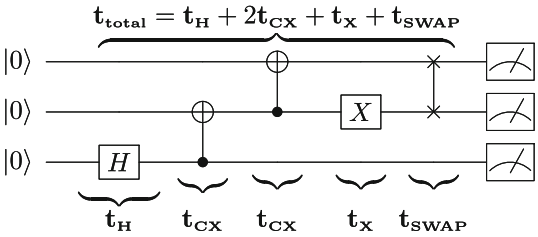


Fig. 1 Example of circuit latency calculation

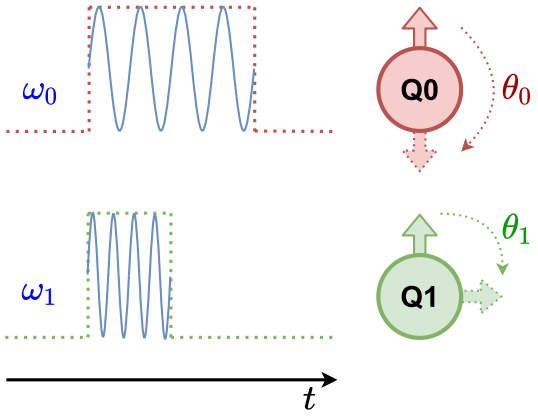


Fig. 2 Correspondence between duration of the field pulse applied to manipulate the spin and the achieved rotation. Assuming the initial state is the up-orientation, a longer pulse leads to a larger rotation angle θ ($\theta_0 > \theta_1$)

executing all the gates of a quantum circuit (Fig. 1 shows an example of circuit latency calculation, assuming that all quantum gates are executed in a strictly sequential way), is fundamental for estimating the circuit execution reliability on a quantum computer. It is important to remind that all quantum technologies, characterized by a qubit encoding derived from a two-level/spin-$\frac{1}{2}$ formalism [7, 8], are affected by dynamic non-ideality phenomena, such as relaxation and decoherence [9], and that the duration of quantum gates is not fixed. For example, single-qubit gates $R_{\{X,Y\}}(\theta)$ are implemented by exciting a two-level/spin-$\frac{1}{2}$ system with a resonant finite-time electromagnetic field, whose **duration** is proportional to the gate rotation amount θ [9, 10], as clear from Fig. 2. In order to keep negligible the effects of undesired phenomena, the circuit latency should be minimized; this can be done at compilation level by finding an equivalent circuit involving quantum gates with minimum duration.

The case of circuit latency optimization puts in evidence the necessity of exploiting figures of merit in quantum compilation taking into account the features of hardware in some way. In classical digital design and synthesis, dimensionless normalized quantities are often employed as cost functions. For example, the maximum delay of a circuit can be evaluated from the normalized execution times of its modules [11], which are estimated considering the logic gates constituting each module. A similar approach could be exploited for quantum circuits; figures of merit, assigning different weights to different gates according to their duration and implementation complexity, have been already proposed. An example of these is the cost function of the Exercise 4 of the IBM Quantum Challenge 2020 [12], related to the construction of the optimal

quantum circuit of a unitary matrix:

$$f = 10 \cdot n_{\mathrm{CX}} + n_{\mathrm{U3}} \,. \tag{1}$$

Parameters n_{U3} and n_{CX} are the total number of U3 (single-qubit) and CX (two-qubit) gates, respectively, each of which has a different weight (1 and 10, respectively). Another example is the cost function discussed in Sect. 4.3 and used in this article, which takes into account the circuit depth and the fidelities of quantum gates, whose definition will be cleared in that section.

Current state-of-the-art compilers adopt different approaches for building the circuit, most of which exploit heuristic methods at their core. In this work, none of the more *mainstream* optimization strategies was chosen. In fact, instead of resorting to algebraic manipulations or graph theory, a more straightforward method was explored to evaluate if a simpler local optimization could favour the technology-specific compilation: a **template-based approach**. This method consists in exploiting circuit equivalences to shuffle the quantum circuit's structure and then obtain optimizations, as represented by the example in Fig. 4 using IBM's Quantum Experience [13]. When analyzing the quantum circuit represented in the left part of Fig. 4b, by employing the circuital equivalence described in Fig. 4a it is possible to create a redundancy to be exploited for the optimization of the circuit as a whole. The result is represented in the right part of Fig. 4b.

The goal of this article is twofold. Firstly, to determine whether a local compilation strategy based on templates could achieve similar (or even better) real-world results as those provided by a significantly more complex global compilation approach. Secondly, to provide a *prototype of a core for a modular quantum circuits compilation toolchain*, to use as an easily expandable and reliable base for future, more competitive developments. All the equivalences are reported in the supplementary document *Avitabile et al., Supplementary Information*. The compilation strategy has been tested on circuits to be executed on three supported quantum computing technologies: nuclear magnetic resonance (NMR) [14], trapped ions [15] and superconducting qubits [16]. These technologies are placed on the same level of attention in the proposed compiler; for this reason, the effort to equalize them can be considered a unique feature of the toolchain, since it differentiates it from other compilers in the state of art, that—even when they support native gates of different technologies—are mainly focused on the more easily accessible superconductive backends (suffice to remind the fewest-qubit freely accessible superconductive quantum computers from IBM).

After a brief overview of the state of the art in quantum Design Automation methods and a summary description of the compilers used in the testing phase in Sect. 2, the in-depth optimization approach, structure and technology-related optimizations of the proposed toolchain are discussed in Sect. 3. In Sect. 4 the benchmark procedures used to test the toolchain performance, in terms of compilation time and complexity of the final circuit, are explained and the obtained results are presented and compared with those of two state-of-the-art compilers, in terms of different weighted and non-weighted figures of merit. Evaluation of the toolchain competitiveness and future perspectives are then discussed in Sect. 5.

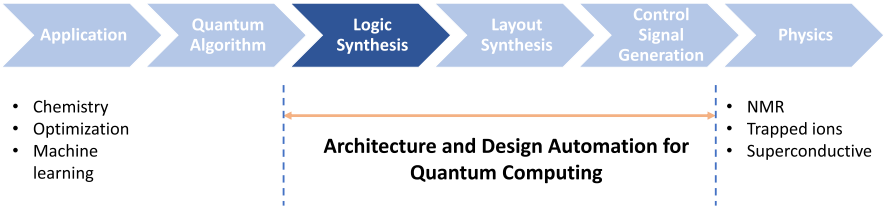


Fig. 3 State-of-the-art quantum architecture and design automation toolchain. The logic synthesis step, on which the current version of the proposed compiler is focused, is highlighted. Image arranged from [17]

2 Background

Taking inspiration from classical computing, the quantum state of the art rapidly adopted the philosophy of the Design Automation to produce reliable architectures. The process of definition of an optimal workflow to design and optimize a quantum processor is still in part a matter of trial and error in development, but in the last decade the comprehension of quantum algorithms and quantum circuits has steadily improved in both industry and academy. Nowadays, the most crucial steps to efficiently optimize a quantum circuit are known, defined and implemented in several state-of-the-art design processes, and this gave birth to a somewhat canonical architecture design toolchain structure, such as the one described in [17] and represented in Fig. 3. As for the design methods adopted in the workflow, as stated before, there are many philosophies employed in the current state of the art. Some prefer to construct **optimal** realizations of reversible circuital structures defined by a given number of inputs whenever possible [18], but in more general case other strategies are commonly devised. Some of the most common approaches are based on **heuristic** methods that rely on a plethora of possible different logic mechanisms (transformations, Binary Decision Diagrams or BDDs, unitary matrices evaluation or search algorithms, such as the A*) [1, 18, 19] to produce reasonable solutions starting from a fixed amount of available computational resources [20]. These methods also have the peculiarity of being one of the few feasible ways to solve NP-hard problems in practice [21]. Other famous optimization strategies employ more advanced **meta-heuristic** algorithms to implement more unique and complex solutions. Furthermore, there is a niche of rare or experimental approaches, such as the template-based strategy [22], which will be discussed more in detail in Sect. 3.

Both the state-of-the-art quantum design flow and improvement strategies play a *behind the scenes* role and are implemented in the inner mechanisms of **quantum software platforms** [23]. The purpose of these software environments is to make available to the user a set of manipulation tools intended for quantum programs, and they are usually in charge of managing the compilation and optimization of quantum circuits in technology-agnostic and technology-specific contexts alike. Several big players in the quantum research field came up with their software platforms, using different programming languages or quantum intermediate representation languages. As references for the comparisons made to evaluate the proposed toolchain performance, two particularly successful state-of-the-art compilers were chosen: **IBM's Qiskit** [24–26]

and **Cambridge Quantum Computer's t|ket⟩** [27, 28]. Before introducing these compilers, it is important to specify that, similarly to the toolchain described in this work, they both handle quantum circuits described in **OpenQASM 2.0** [29], an intermediate representation language proposed by IBM in 2017 for formalizing the design of circuits to be executed on its hardware through the Quantum Experience.

Qiskit is an open-source framework for quantum computing developed by **IBM Research** and implemented in **Python** language. The tool set specifically used to generate comparison circuits analyzed in Sect. 4 was the **Qiskit Terra Transpiler**, which allows an optimized transpiling of quantum circuits using a user-defined set of gates. The Transpiler supports four **optimization levels**, named 0, 1, 2 and 3, where an higher level corresponds to a finer optimization. Qiskit has been widely used in the quantum research community for years. Indeed, IBM's policy of open-sourcing it and of making available tools like Quantum Experience for free made it one of the most employed quantum compilers, especially when dealing with superconducting devices, which are IBM's flagship quantum technology.

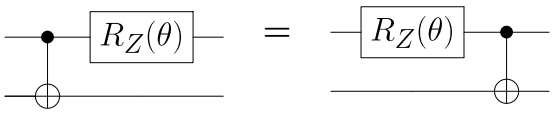
The second reference compiler, **t|ket⟩**, is an architecture-agnostic quantum software developed by **Cambridge Quantum Computing**, implemented in **C++** language. Originally this was a closed-source software, but Cambridge Quantum Computing decided to make it open-source recently [30]. This compiler manages the transpiling of machine-independent quantum algorithms into optimized quantum circuits, and it supports multiple intermediate languages (including OpenQASM 2.0). It also allows two optimization levels: **standard and maximum**. t|ket⟩ was closed-source at the time of development of the work described in this article and a Python module called **pytket-qiskit** was employed to access it and interface it with Qiskit for generating the reference circuits. In the past few years, t|ket⟩ proved to be an efficient platform in the state of the art, outperforming most competitors in several published benchmarks. It is especially renowned for its capability to smartly adapt a circuit to a given target device and to manage multi-qubit gates.

These two state-of-the-art compilers employ different optimization strategies. **Qiskit** uses a custom, internally defined sequence of algorithms and converters that employ on **DoCPLEX** [31], an IBM optimization library based on decision optimization through prescriptive analyitics [32]. t|ket⟩ optimization strategy, as reported in [33], consists in a combination of "Peephole optimizations" (based on the identification of certain patterns in small windows inside the circuit), of algebraic Euler and KAK [34] decompositions (which are a known compaction method used for QCs) and of mathematical optimizations of high-level macroscopic structures. Most of these optimization methods are on average more complex than the template-based scan performed by the toolchain.

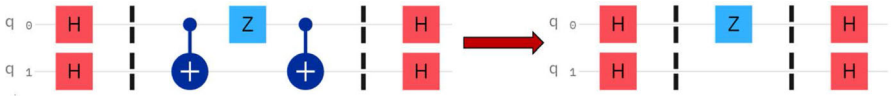
3 The proposed toolchain

3.1 The template-based approach

As introduced in Sect. 1, during the development of the toolchain instead of resorting to a complex optimization strategy based on algebraic evaluations or branching



(a) Exploited equivalence.



(b) Application of the equivalence to reduce the complexity of the circuit.

Fig. 4 Example of application of a template as described in [35] to perform a circuital optimization

diagrams, the more "circuital" **template-based approach** was explored. According to this technique, the input quantum circuit is scanned, and a series of circuit equivalences and identities, denoted as **templates** and described in the toolchain's libraries, are identified. Once a template is detected, the quantum gates adjacent to it are identified and, if deemed convenient, the template structure is "switched" to its equivalent form to obtain a compaction or optimization in the circuit (as represented in Fig. 4). This approach is similar to the "Peephole optimizations" employed by t|ket⟩ [33], and it is based on the very simple and intuitive concept of *circuital equivalences* that does not require complex mathematics or algebraic evaluation tools to be implemented. The intrinsic flaw of this philosophy is that it suffers from a **limited foreseeing**, since it is based on case-by-case applications performed while scanning the circuit, and is incapable of looking for multiple steps ahead. Moreover, since the quantum circuits are optimized by applying a remodeling based on a purely circuital evaluation, without analyzing the exact state of given qubits after or before each operation, the template-based approach does not consider all optimizations or equivalences based on knowing the state of certain qubits, favouring general optimizations instead. The only employed state-dependent optimizations involve the elimination of eventual R_Z gates at the beginning of each qubit line, which is assumed to be initialized to $|0\rangle$ (since $R_Z|0\rangle = |0\rangle$), or before a measurement (since it does not affect the probability distribution of eigenstates). Also, it has to be noted that, in the purpose of applying this core philosophy in the toolchain's workflow, many implemented optimizations tend to prefer R_Z gates over other quantum gates. This is based on the staple principle that R_Z gates can be **implemented virtually** [10, 36] in an advantageous way that does not hamper the circuit latency.

Currently, the toolchain's libraries contain a total of **27 templates** belonging to *three families* (all reported in *Avitabile et al., Supplementary Information*). These templates are far more numerous t|ket⟩'s 7 "Peephole" identities [33], but less than **Qiskit**'s 71 template circuits [37]. However, it has to be noted that a direct comparison between the toolchain's templates and the Qiskit ones cannot be properly done.

First of all, Qiskit employs a broader definition of "template", which is a sequence of gates providing the identity as global evolution. This implies that the simple sequences

of two identical gates, with $U^\dagger = U$ (such as H, X, Y, Z, CX and CZ) are properly treated as templates. On the other hand, templates of the presented toolchain are equivalent sequences of quantum gates, "topologically" different from each other (in sense that $U^\dagger \neq U$), which can be exploited for achieving a reduction of the total number of involved gates in a quantum circuit. This definition is a bit more strict and does not involve the previously mentioned sequences where $U^\dagger = U$, which are in any case removed in a circuit by the toolchain because of their intrinsic simplicity.

Another reason of the difficulties of a quantitative comparison between the templates of Qiskit and those in the toolchain is that the first one involves forty-seven templates with Toffoli gates, which are currently too hard for a direct hardware implementation and for this reason they are not involved in the toolchain, operating with gates involving at most two qubits, more suitable for experimental execution.

The described prototype classifies templates structures in three groups:

1. *Simple gate equivalences* (*SGE*), 15 templates reported in Section 2.1 of the supplementary file. They involve only single-qubit gates. Figure 7b shows equivalences belonging to this family.
2. *Templates that involve single-qubit gates and One Two-Qubit gate* (*T1TQ*), 8 templates reported in Section 2.2 of the supplementary file. Some of these templates involve two different forms to be exploited as circuital equivalences, but for naming and counting purposes, those templates were still considered as a single instance instead of two different templates. In all these templates, the two-qubit gates are CX or CZ. A template belonging to this family is represented in Fig. 4a.
3. *Templates acting on clusters of multiple two-qubit gates* (*TCTQ*), 4 templates reported in Section 2.3 of the supplementary file. Circuits with multiple two-qubit gates, such as the one represented in Fig. 11a, belong to this family.

There are also a few special templates that apply to the trapped ion technology's own two-qubit gate $\mathbf{R_{XX}}$ and to NMR technology's own two-qubit gate $\mathbf{R_{ZZ}}$ (reported in Section 3.2 of *Avitabile et al., Supplementary Information*) that for counting and nomenclature purposes were treated as an independent category.

The templates and identities contained in the toolchain's libraries were extrapolated and validated from [8, 38–42] or obtained through calculation and matching of mathematical identities. These equivalences can be both technology-agnostic—i.e., applicable to all circuital structures and do not employ any quantum gate specific of a given target technology—and technology-specific, *i.e.,* focused on creating more advantageous circuital structures for a given target technology, taking into account its native gates. The focus on the latter kind of optimization is a peculiarity which sets apart the toolchain from most state-of-the-art compilers. Instead of adopting an optimization philosophy which revolves around a single technology (usually the simpler superconducting one) and then performing a final translation and post-processing to adapt the output circuit to other target technologies, the toolchain evaluates during its run time *specific optimizations* based on the supported technologies' own library of intrinsic gates. This is the case, for example, when it smartly manages **CZ gates** when optimizing the circuit for a technology that does not directly support them, or when considering the application of Special Templates right after the translation of CX gates into specific two-qubit gates has been performed (as described in Sect. 3.2).

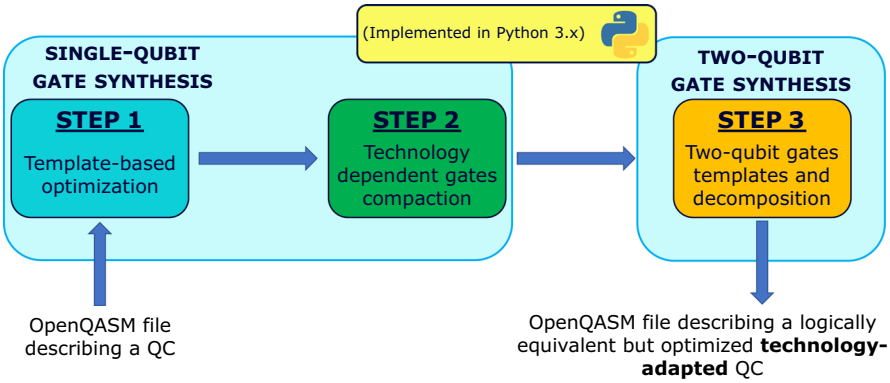


Fig. 5 Overall workflow structure of the proposed toolchain. Each **Step** operates a specific subset of manipulations, and has its own *ad-hoc* inner morphology in order to optimize the input circuit as much as possible

3.2 The toolchain's structure

The proposed quantum toolchain works by taking as input a given quantum circuit described in OpenQASM 2.0 and producing as output an optimized circuit, decomposed using the set of gates of a target technology, also described in OpenQASM 2.0. Referring to the state-of-the-art toolchain described in [17] and represented in Fig. 3, the toolchain operates by implementing the optimizations proper of the *Logic Synthesis* by using a sequence of three different scripts labeled as **"Steps"**. The overall workflow is represented in Fig. 5.

The optimizations performed by the toolchain are applied step-by-step on an incrementally specific target. In *Step 1*, the whole **Clifford + T gate set** is targeted by technology-agnostic optimizations with a focus on single-qubit gates, while the application of specific templates for multi-qubit gates are delegated to a later step. In *Step 2*, only the *technology-dependent gate set* is taken into account, so all single-qubit gates are translated into their corresponding of the target technology's native set and then optimized accordingly; clearly, to perform these operations, *Step 2* forsakes the technology-agnosticism of Step 1 to employ a **technology-specific approach**, taking in consideration only the chosen target technology and its inherent optimization process. Finally, in *Step 3* an *ad-hoc* set of optimizations is applied to all **multi-qubit gates** through the application of appropriate templates and decomposition based on the target technology. Some of the Steps feature some inner loops, (e.g., those associated with the parameters `IT1` and `IT2` in Fig. 6) to allow a tunable grade of circuit compaction through the reiterate application of a certain subset of templates.

3.2.1 Single-qubit gates synthesis block

As observable in Fig. 5, the single-qubit gates synthesis block involves the first two steps of the toolchain:

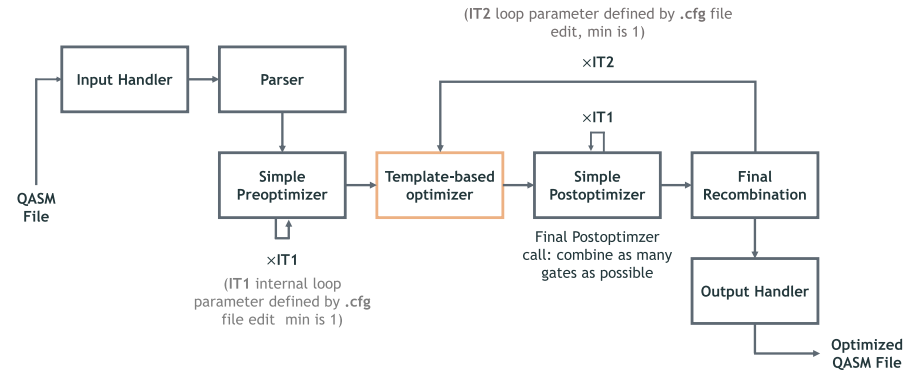


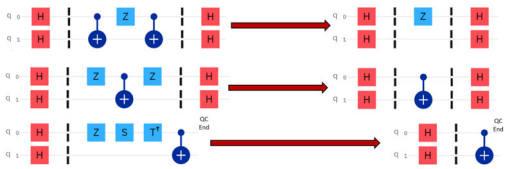
Fig. 6 Toolchain Step 1's in-depth workflow

- *Step 1: QASM template-based optimization (Fig. 6)*—The aim of this step is to apply the main bulk of the circuital equivalences and template-based substitutions described in the toolchain's libraries and to compact the input circuit as much as possible, as represented in Fig. 7a. This is achieved through the reiterative application of a series of "coarse" compactions (performed in the **simple preoptimizer** and **simple postoptimizer** blocks in Fig. 6 and based on the templates belonging to the SGE family) followed by specific, fine-grain remodelings (performed in the **template-based optimizer** block in Fig. 6 and based on the use of the T1TQ templates, which are applied to generate as many circuital null operations as possible, without hampering the logic of the circuit). When a straightforward elimination of redundant gates is not feasible, this Step implements some transformations to maximize the use of preferable kinds of gates, such as the virtually implementable R_Z gates (in this case, all equivalences pertaining R_Z gates from the ones of the SGE family). Step 1 is designed to be completely **technology-agnostic** and its optimizations are particularly efficient in reducing the number of single-qubit gates in the circuit. It exploits templates belonging not only to the SGE family, but also to the T1TQ one, such as the one represented in Fig. 4a, thus allowing a situational and yet efficient set of circuital improvements. As of now, Step 1 is compatible with an extended *Clifford* + *T* gate set, and it supports the usage of $\boldsymbol{R_X}$, $\boldsymbol{R_Y}$, $\boldsymbol{R_Z}$, **X**, **Y**, **Z**, **S**, **T**, $\boldsymbol{S^\dagger}$, $\boldsymbol{T^\dagger}$, **H**, **CX**, **CZ** and **CCX** (or **Toffoli**) gates in the input quantum circuit. IBM's gate set comprising $\mathbf{U1}(\lambda) = R_Z(\lambda)$, $\mathbf{U2}(\phi, \lambda) = R_Z(\phi) R_Y\left(\frac{\pi}{2}\right) R_Z(\lambda)$ and $\mathbf{U3}(\theta, \phi, \lambda) = R_Z(\phi) R_X\left(-\frac{\pi}{2}\right) R_Z(\theta) R_X\left(\frac{\pi}{2}\right) R_Z(\lambda)$ gates is also passively supported, but no optimizations are performed on such gates until Step 2, since it is assumed that the original input circuit only employs non technology-specific quantum gates. Generally speaking, the output produced by this Step is an optimized OpenQASM-described circuit in which all gates have been decomposed to the $\boldsymbol{R_X}$, $\boldsymbol{R_Y}$, $\boldsymbol{R_Z}$, **CX**, **CZ** subset of gates, and in which each Pauli gate is transformed in its rotational form using floating point notation. Step 1 accepts as input a **Subcircuit parameter**, which is a boolean flag that defines if the circuit is indeed a *subcircuit* to be used in conjunction with other QASM-described entities and thus if the optimization regarding R_Z gates at the beginning or the end of a

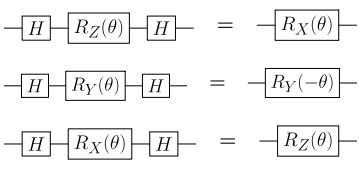
circuit can be employed.
For what concerns the Step 1 matching order, the application of simple identities in the circuit is always performed preliminarily to the application of more complex templates. During this process, the toolchain evaluates all gates that could be a part of a complex template and leaves them "untouched", while compacting the others. As for the complex templates, the matching is performed in a custom but interchangeable order, since all templates do not overlap between themselves. The only exception is a template concerning H gates (reported as Template H1 in Section 2.2 of *Avitabile et al., Supplementary Information*) which is a particularly convenient subcase of a more general template (reported as Template H2 in Section 2.2 of *Avitabile et al., Supplementary Information*); in particular, Template H1 is applied before Template H2 to ease the detection of the subcase structure and to speed up its substitution. Overall, the Step 1 makes extensive use of reiterated customizable loops to ensure the maximum grade of circuital compaction is achieved, as represented in Fig. 6.

- *Step 2: Technology-dependent gates compaction (Fig. 8)*—The aim of this step is to translate the output circuit of Step 1 into the proper set of gates relative to the target technology, which could be specified as input and chosen from the **nuclear magnetic resonance (NMR), trapped ions and superconducting technologies**. The translation process, which is applied universally to single-qubit gates, does not implement a decomposition of two-qubit gates, such as the CX gates, which are left untouched in order to be exploited once in Step 3, where they will eventually be decomposed into their basic constituting gates. This procrastination in the workflow is due to the fact that all the templates that involve multiple two-qubit gates can be detected and performed much more easily with non-decomposed CX and CZ gates. This step is designed to take as inputs the optimized circuit *.qasm* files generated by Step 1 to work at maximum efficiency, but it can also be used on custom, unoptimized *.qasm* files. Along with this translation, Step 2 employs a manipulation on triplets of adjacent rotation gates of different type to achieve a further compaction of the circuit and to increase the number of R_Z gates when possible, based on the toolchain's assumption that they can be implemented with a null duration and are thus advantageous to maximize.
  This manipulation, named **Eulercombo**, is quite powerful, and it is based on coordinate transformations using ***Euler angles*** [43], as represented in Fig. 9. Basically, it scans the circuit and identifies triplets of consecutive single-qubit gates: once a triplet is detected, it identifies its adjacent gates and evaluates the most convenient coordinate transformations to manipulate the triplet in a way to change its external gates in the same type of their adjacent gates, thus allowing a compaction. When multi-qubit gates are involved, Eulercombo tries to generate a favourable template with them, and if this is accomplished then it manages an exploitation localized on that circuit subsection. As showed in Fig. 8, Step 2 also features:

  – A smart disposing of CZ gates for the technologies that do not support them, featuring a rearrangement in order to exploit the symmetry property of CZ gates in order to maximize the null operations, followed by an *ad-hoc* translation into



(a) Some circuital optimizations based on the template in Figure 4a.



(b) Some Hadamard-based single-qubit equivalences.

Fig. 7 Examples of circuits compaction achievable with the available templates

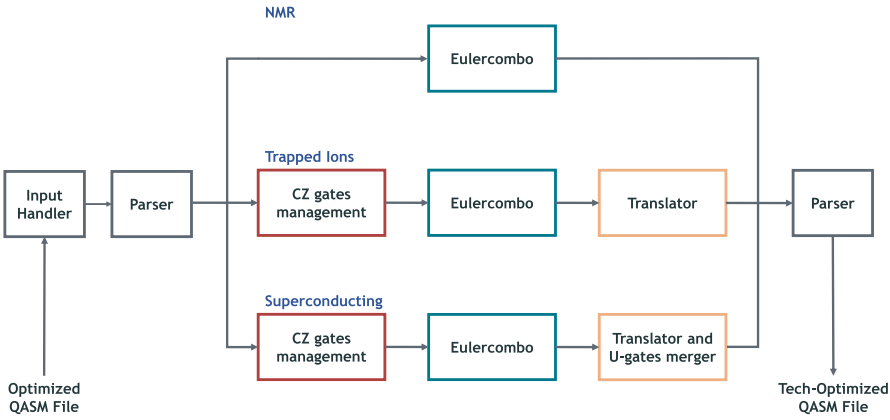


Fig. 8 Toolchain Step 2's in-depth workflow

CX gates using the equivalences reported in Figure 2 in Section 1 of *Avitabile et al., Supplementary Information*.

- The optimized merging scheme for IBM's U gates set proposed in [44], that uses the same core philosophy of the Eulercombo to maximize the compaction of U3 gates.
- A series of further circuital compactions based on simple translations into preferable gates, such as the R_Z gates for NMR and trapped ions technologies and the U2 for the superconducting technology.

While Step 2's role in the optimization process is fundamental to obtain quantum circuits that are tailored to a specific implementation technology, its optimizations are mostly situational, when compared to Step 1's powerful set of circuit improvements. In terms of circuit optimization, Step 2 *consolidates* the reduction of the total number of single-qubit gates and it is particularly effective in dealing with long streaks of these kind of gates uninterrupted by two-qubit gates. As of now, Step 2 supports the usage of $\boldsymbol{R_X}$, $\boldsymbol{R_Y}$, $\boldsymbol{R_Z}$, **CX** and **CZ** gates in the input quantum circuit and requires all single-qubit gates to be adjacent to gates of different type. The U gates set is supported in the case of the superconducting target technology, but not in the others, where an extra step of decomposition into R_X, R_Y and R_Z gates (not implemented in the current toolchain) is required. This is due to the core assumption that input circuits do not employ technology-specific gate sets,

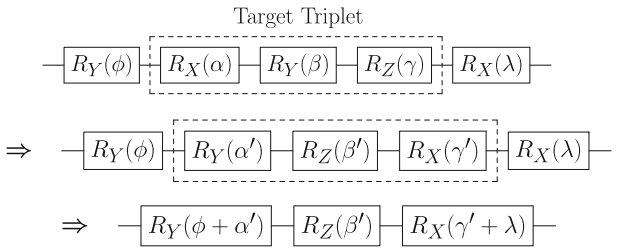


Fig. 9 Example of an Eulercombo application performed in Step 2. The triplet of gates in the middle is rearranged using Euler angles in order to allow a recombination of both gates on the sides and thus a compaction of the circuit. The usage of a central R_Z gate is preferred since none of the other two external gates is of the same type, as explained in Sect. 3.1

just like in Step 1. All the manipulations involving *Euler angles* are performed through the usage of the **SciPy Python library** and of the **NumPy Python library**. Step 2 also accepts as inputs both the Subcircuit parameter and a parameter used to define the target technology. The toolchain supports the usage of the following gate sets depending on the target technology:

- **NMR technology: R_X, R_Y, R_Z, CX and CZ gates**.
- **Trapped ions technology: R_X, R_Y or R(θ, ϕ) gates, R_Z, CX gates**.
- **Superconducting technology: U1, U2, U3 and CX gates**.

3.2.2 Two-qubit gates synthesis block

- *Step 3: Distribution/mirroring-based optimizations and CX gates decomposition (Fig. 10)*—The aim of this step is double: the exploitation of a certain subset of templates that may ensure a steady reduction of the number of employed CX gates (using the TCTQ templates, reported in Section 2.3 of *Avitabile et al., Supplementary Information*), and the decomposition of each two-qubit gate by using the target technology's own native library. This step currently does not cover the effective operations needed to the layout synthesis block described in the state-of-the-art toolchain in [17], since it lacks a routing mechanism capable of taking into account a given device's connectivity and to map a circuit accordingly. For this first prototype of toolchain, in fact, it was preferred to focus on general-purpose optimizations and on the adaptation to theoretically fully connected technologies. Moreover, inserting SWAP gates to match the circuit's usage of multi-qubit gate with a non-fully connected target device is left as a potential future evolution of the project. This step is designed to take as inputs the optimized circuit *.qasm* files generated by Step 2 to work at maximum efficiency, but it can also be used on custom, unoptimized *.qasm* files. This step is **technology-specific** exactly like Step 2, and the performed manipulations of the circuit differ greatly depending on the target technology. Step 3 has the important task of handling the complex templates that involve clusters of **CX gates**, such as the one represented in Fig. 11a, and is the toolchain's primary source of *multi-qubit gates optimizations*. This task is followed by another important one: a smart **decomposition of two-qubit gates**

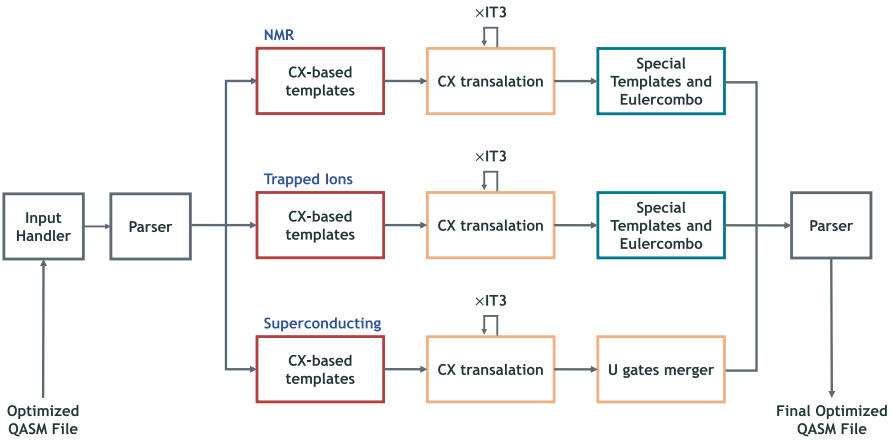


Fig. 10 Toolchain Step 3's in-depth workflow

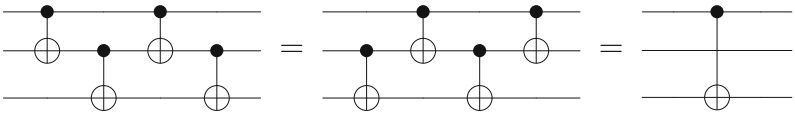
(referred to as *Special Templates* in Fig. 10) and reported in Section 3.2 of *Avitabile et al., Supplementary Information*, aimed at minimizing the resulting translated and newly inserted single-qubit gates' rotation angle and thus the overall circuit latency followed by some reiterated minor optimizations (including Eulercombo calls), all in order to fully adapt the circuit to the target technology with the least impact on the circuits' gates, such as represented in Fig. 11b. In the superconducting case, where CX gates are not decomposed, this task focuses on ensuring the optimal compaction scheme proposed in [44] on all existing U gates (this is performed in the **U gates merger** block in Fig. 10). Both roles performed by Step 3 are essential to complete the toolchain's proposed compilation process. This step requires all single-qubit gates to be adjacent to gates of different type. Step 3 also accepts as inputs both the **subcircuit parameter** and a parameter used to define the **target technology**. When processing an input file generated from the Step 2, it is capable of automatically recognizing the used target technology without needing an input.

As represented in Fig. 10, the technology-specific templates are always applied after the technology-agnostic ones, which are also applied in a reiterated manner to achieve an higher grade of circuit compaction.

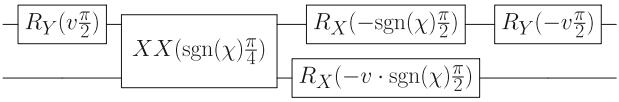
At the end of the compilation phase, circuits compiled for each examined technologies are characterized by the following gates:

- **NMR technology: R_X, R_Y, R_Z and R_{ZZ}.**
- **Trapped ions technology: R_X, R_Y or R(θ, ϕ) gates, R_Z and R_{XX}.**
- **Superconducting technology: U1, U2, U3 and CX gates.**

While in the compilation of circuits for trapped ion and NMR technologies the effective native unitary evolutions are employed, the circuit compilation for superconducting qubits employs the CX, which is not native of this technology, but can be built from its effective two-qubit unitary evolutions [45]. According to the qubits functioning, the characteristic gate can be either the cross-resonance—substantially a $\mathbf{R_{ZX}}$, exploited



(a) Example of a CX cluster-based template exploited in Step 3 to greatly reduce the number of multi-qubit gates whenever the right distribution is detected in the circuit.



(b) Example of decomposition of a base CX gate in Trapped Ions technology by using the XX gate, in which v is an arbitrary parameter with the value of ∓ 1, as shown in [48].

Fig. 11 Overview of operations executed in Step [64]

by devices with fixed resonance frequency—or the Controlled-phase or the iSWAP, typical of flux-tunable qubits.

In the proposed toolchain, the *abstract* gate set for IBM superconducting qubits, available in the Qiskit transpiler at the time of development of toolchain itself, is employed. This choice was made for two reasons: the first one is that the Qiskit transpiler was expected to be, even before the development of the toolchain, one of the references to be taken into account for the comparative evaluation of the presented work. The second one is that, in the compilations for all available technologies, gates which were fully supported by the QASM simulator available in Qiskit were chosen in order to facilitate the functional verification of the compiled circuits, and CX belonged to this gates set.

3.3 Implementation overview

The toolchain was implemented using a sequence of **Python 3.x scripts**. The choice of Python as the programming language used to build the current version of the prototype is due to the necessity to interface it with existing quantum computing frameworks, and in particular with their quantum circuits simulators, thus permitting to benchmark and validate template-based compilation with ease. Each of the three main scripts implements one of the three Steps and takes as input an OpenQASM 2.0-described *.qasm* file. To ensure the complete application of the toolchain as intended, each step must take as input the file produced as output by the previous step or, in Step 1's case, the original quantum circuit file to be optimized. It is also possible to apply the specific optimization described into one of the three main scripts to an *ad-hoc*, custom file. In fact, even though a one-step technology-specific approach might lead to slightly better results for a single target technology, the whole toolchain was intentionally designed in a completely **modular** way through a **libraries-based implementation**, which allows the possibility of dealing with several technologies, total control on which steps are applied to a certain circuit, high flexibility in the functions' usage, the capability to

set some specific parameters through the edit of specific files and the faculty to allow future modifications and expansions, such as the integration of novel technologies for quantum computers, like semiconductor quantum dots [46], defects in diamond [47], *et cetera*. Following the philosophy of total modularity, several libraries in form of scripts were created, each containing a subset of functions designed to tackle a certain specific part of the optimization process. This allows the main script for each Step to remain well-ordered and easily customizable, and it also facilitates the nested usage of the functions in multiple occurrences, while making each library easily readable. In order to allow the edit of certain parameters that are particularly uncomfortable to pass as shell inputs, each step supports the reading of **.cfg files** to determine such parameters and to act accordingly. These parameters include the grade of approximation of π when dealing with rotational gates, the threshold of rotation value under which a gate is considered a null operation, the iterative parameters used to arrange the function loops in the workflow and other technology-related flags.

4 Benchmarks

4.1 Testing methodology

To test the circuits generated by the toolchain's steps, some benchmark scripts were created in **Python language**. The first mandatory aim of these testing scripts was to verify that the introduced optimizations were actually correct, and that the outcome of each circuit was the same as the reference quantum circuit. To do so, both the *.qasm* file describing the reference quantum circuit and the *.qasm* files generated by the toolchain were used to create quantum circuits in **IBM's Qiskit** and to simulate them with the **QASM Simulator** available in the **Aer Library**, thus permitting the verification of their equivalence.

QASM Simulator is the most consolidated simulator in Qiskit for simulating noisy or large and deep quantum circuits. One of its peculiarities is that, when a measurement operation is called for the circuit to be simulated, it returns a measurement counts distribution for the eigenstates, depending on the number of shots N_{shots}, instead of the exact eigenstates probability distribution. In other words, QASM Simulator provides an N_{shots} "finite estimation" of the probability distribution, which would be obtained in the limit $N_{\text{shots}} \to \infty$. From a practical point of view, N_{shots} should be higher for non-uniform distributions with most of eigenstates having non-null probabilities of being measured, such as in the **ISING_N10** circuit, reported in Tables 2, 3 and 4, with mean probability value slightly lower than $1 \cdot 10^{-3}$ and maximum one equal to $42 \cdot 10^{-3}$

By using the simulator's standard settings, with N_{shots} depending on the circuit to be simulated, and by ensuring that every qubit line in each tested circuit had a final measurement performed, it was possible to simulate each circuit and to visualize the corresponding eigenstates measurement counts in form of an histogram. In case of circuits where a single output state was expected with a probability of 100%, the relative optimized circuit was deemed as "correct" if the output matched completely with a probability of also 100%. In case of circuits in which multiple output states with

different probabilities were expected, the **Kullback–Leibler deviation** [48] between the obtained results was computed using the following formula, in which N is the total number of output states with non-null probability, opt(i) and ref(i) are the occurence of the *i-th* eigenstate in the optimized and reference circuit, respectively, and N_{shots} is the total number of measurement shots associated with the employed simulator:

$$\text{KLD} = \sum_{i=1}^{N} \frac{\text{opt(i)}}{N_{\text{shots}}} \cdot \log_2 \left(\frac{\text{opt(i)}}{\text{ref(i)}} \right) \tag{2}$$

$N_{\text{shots}} = 8192$ was deemed as sufficient in all tests, except for **ISING_N10**, where it was set equal to 500000. The obtained results were then marked as "correct" only if the deviation did not exceed a value of $5 \cdot 10^{-3}$. Once the optimized circuits were verified as "correct", the scripts' aim became to evaluate the effective capabilities of the toolchain. Evaluation was done in terms of estimation of the complexity of the compilation algorithm, in terms of execution time, and of comparison of its compiled circuits with those obtained with the compilers in state of the art, in terms of different figures of merit. All the results are available in Sects. 4.2 and 4.3.

4.2 Estimation of complexity of the compilation algorithm

In order to estimate the complexity of the template-based compilation algorithm, a random-tests-based evaluation of the compilation time has been performed by changing the number of involved qubits and by considering all the target technologies currently supported by the toolchain.

First of all, for each qubits parallelism, 100 dense random unitary matrices were generated, then the equivalent quantum circuit of each matrix—based on the extended *Clifford + T* gate set discussed in Sect. 3.2.1—was obtained with Qiskit and subsequently compiled on the toolchain for supeconducting, trapped ions and NMR technologies. Repeated tests permitted to compute a mean of the compilation time for each qubits parallelism and each target technology.

These tests were done to stress the toolchain in a **worst-case scenario**, in terms of both total number of executed operations and compilation time. In fact, the choice of exploiting random matrices is due to the necessity of verifying the eventual efficiency limits of the toolchain in an unpredictable operating regime, where the presence of template-based optimizations is not known *a priori*. Moreover—considering that quantum circuits generated from random matrices have a total number of gates that is definitively higher than the corresponding ones of application-specific quantum circuits (such as those in Tables 2, 3 and 4)—the compilation times are expected to be definitively longer for the same number of qubits. It has been observed that, for each qubits parallelism, Qiskit generates quantum circuits from random unitary matrices with the same number of gates and order of single and two-qubit gates, thus implying that the same decomposition strategy is adopted. The number of gates for qubits parallelism from 2 to 6 are reported in Table 1. Another option forcing a worst-case computational scenario is the choice of *Low precision*, since its "looser" criteria

Table 1 Number of gates with (without) measurements of quantum circuits associated with random unitary matrices

Number of qubits	Number of gates
2	77 (75)
3	216 (213)
4	1020 (1016)
5	4507 (4502)
6	18964 (18958)

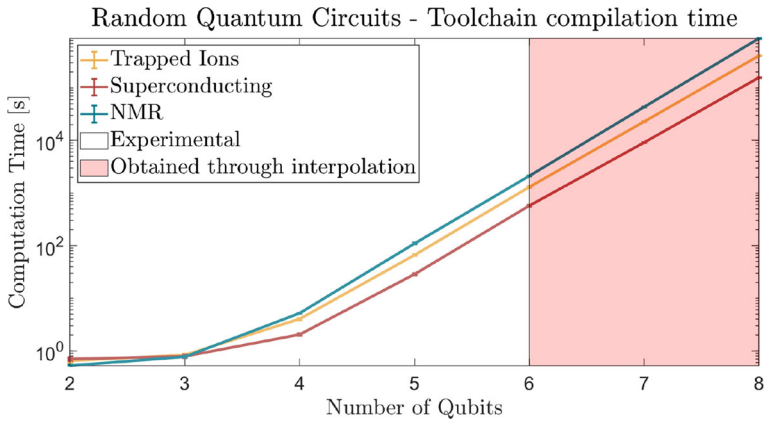


Fig. 12 Execution time for completing compilation of random circuits with different parallelisms and target technologies

brings to an higher number of allowed recombinations and, thus, to more performed evaluations and operations and to an higher computation time.

The left part (white background) of Fig. 12 shows the obtained results for a number of qubits between two and six. A similar exponential trend is visible in the three cases, in particular for a number of qubits between four and six. This implies that the compiler behaves in a coherent way for all the examined technologies, at least for what concerns the number of operations to be done, which is reflected in the total compilation time. In order to further investigate this similarity, fitting calculations were done, for estimating for each technology a relationship between compilation time Δt and number of qubits N of type

$$\Delta t(N) = \alpha e^{\beta N} \tag{3}$$

and in particular the characteristic multiplicative coefficient of the exponential α. Considering that $\log(\Delta t) = \log(\alpha) + \beta N$, linear fitting was executed for $N \in [4, 6]$ and the following β coefficients were obtained for the three technologies:

$$\begin{aligned} \beta_{\mathrm{S}} &= 2.83 \\ \beta_{\mathrm{I}} &= 2.89 \\ \beta_{\mathrm{M}} &= 3.00\,. \end{aligned} \tag{4}$$

The obtained values can be reasonably considered a satisfactory proof of similarity of the compilation procedure in the three cases. In fact, β_{M}, associated with NMR compilation, is slightly higher (6.4% and 4.1%) than β_{S} and β_{I}, characteristic of the plots of superconducting and trapped ions, respectively. Moreover, β_{I} is about 0.7% lower than the mean value of β, that is equal to 2.91, thus further proving that the exponential coefficient of the current implementation of the compiler is expected to be close to 2.90. This value is in any case quite high, as it is possible to see in the right part (red background) of the plot in Fig. 12. In fact, with $\beta \simeq 2.90$, the compilation timescales of seven-qubit and eight-qubit circuits are $\sim 1 \times 10^4$s and $\sim 1 \times 10^5$s, respectively.

In order to understand the reasons for which superconducting and NMR technologies provided the best and worst compilation time results, respectively, it is important to precise that Qiskit almost always preferred the use of CX gates for the creation of quantum circuits from random unitary matrices. This two-qubit gate is native for superconducting technology, so two-qubit-gate decompositions—according to the decomposition schemes reported in *Avitabile et al., Supplementary Information*—is not required. Hence, new single-qubit gates are not inserted and any additional circuit re-arrangement is required. On the other hand, in NMR's case, all CX gates must be translated in CZ (refer to Figure 2 in *Avitabile et al., Supplementary Information*), with the consequent insertion of extra single-qubit gates and additional circuital manipulations. It is expected that NMR could provide lower compilation times with an input circuit with more CZ gates than CX.

In conclusions, according to the obtained results, it is possible to roughly write the total compilation time as:

$$\begin{aligned} \Delta t(N) &= \alpha e^{\beta N} \\ &= \alpha e^{\beta_{\text{intrinsic}} N} \cdot e^{\delta_{\text{two-qubit}} N} \cdot e^{\delta_{\text{re-arrange}} N} \\ &= \alpha e^{(\beta_{\text{intrinsic}} + \delta_{\text{two-qubit}} + \delta_{\text{re-arrange}}) N}, \end{aligned} \tag{5}$$

where $\beta_{\text{intrinsic}}$ is the exponential coefficient of an "intrinsic" compilation time, associated with all operations done before the eventual two-qubit-gate decomposition, while $\delta_{\text{two-qubit}}$ and $\delta_{\text{re-arrange}}$ are the coefficients of the overhead contributions due to two-qubit gates decomposition and final re-arrangement, respectively.

The exponential increase of the compilation time over the number of qubits casts some doubts on the potential scalability of the present prototype of the toolchain, i.e., on its capabilities of handling quantum circuits with an high number of qubits. Even though this is undoubtedly an important aspect to take into account in the development of classical software assisting quantum computation, it must be remarked that the optimization of the compilation time was not the leading concern of the proposed work. In fact, the latter mainly aims to assess whether a local optimization strategy could assist technology-specific compilation, leading to a reduction of the number of quantum gates, which is a critical task of the current NISQ era. In fact, this makes the difference between a circuit that can actually run on real hardware and one that does not, because of poor resulting fidelity associated with the emergence of dynamic non-ideality phenomena.

In this regard, it must be also highlighted that the current implementation of the compiler is limited, at least for what concerns the execution time, by the employed programming language (Python, which is intrinsically slower than other compiled languages, such as C/C++ [49]) and by the strictly sequential execution of the operations. The last one is a drawback due to an absence of effort profused in the prototype to try to overcome an intrinsic computational limitation of template-based optimizations. As it has been reminded, these revolve around gate-by-gate analysis and operations, often nested in recursive procedures, to be called whenever an improvement can be performed to maximize circuital compaction. It is clear that larger circuits require more recursions and longer compilation times, which could not be compensated by a strictly sequential execution. Parallelizing the compilation can represent a valuable tool to achieve a reduction of the compilation time. This could be done by partitioning the circuit in smaller "independent" regions through barriers—such as those in Figs. 4a and 7a—to be optimized in parallel, *e.g.,* by exploiting multi-threading.

4.3 Comparison with other compilers

The comparison of the results provided by the proposed compilation toolchain and the ones generated by the other state-of-the-art compilers chosen as reference was done in terms of different figures of merit, which are reported in the following (each one with a label reported in parentheses):

- The **total number of single-qubit gates (1Q)**.
- The **total number of non-R_Z/U1 single-qubit gates (1QNZ)**, which are the *de facto* relevant single-qubit gates, since they have a non-null duration, in accordance to the assumption of a virtual implementation of R_Z gates.
- The **total number of multi-qubit gates (MQ)**.
- The **normalized weighted circuit latency for single-qubit gates (L)**, given by:

$$\mathrm{L} = \sum_{R \in \mathrm{1QNZ}} \frac{|\theta_R|}{\frac{\pi}{2}} = \sum_{R \in \mathrm{1QNZ}} 2\frac{|\theta_R|}{\pi}, \tag{6}$$

  i.e., the sum of the normalized durations of single-qubit gates R different from R_Z ones, assumed to be implemented virtually. Since the latency introduced by an $R_{\{X,Y\}}$ gate is proportional to the rotation angle $\boldsymbol{\theta_R}$, a normalized duration $\mathbf{2}\frac{|\boldsymbol{\theta_R}|}{\boldsymbol{\pi}}$ was chosen, so that gates with rotation angles equal to $\frac{\pi}{2}$ and π provide circuit latency contributions equal to 1 and 2, respectively. In the case of the quantum circuit in Fig. 1, since H and X gates are both usually translated into one non-virtual single gate with $|\theta_R|$ equal to $\frac{\pi}{2}$ and π, respectively, [10], $\mathrm{L} = 1 + 2 = 3$. These weights can be used not only for technologies based on $R_{\{X,Y\}}$ gates, but also with IBM's U gates. In fact, considering the U-R gates relations reported in Sect. 3.2.1, U2 and U3 gates are characterized by one and two non-virtual gates, respectively (all with $|\theta_R| = \frac{\pi}{2}$), so their corresponding circuit latency contributions are equal to 1 and 2.
- The **circuit cost C**, that, according to [50], can be computed as follows:

$$\mathrm{C} = -D \log K - \sum_i \log \mathcal{F}_i^{1q} - \sum_j \log \mathcal{F}_j^{2q} \,, \tag{7}$$

where D is the circuit depth, K is a constant that increases the cost of deep circuits, $\mathcal{F}_i^{1q}$ is the average fidelity of the i-th single-qubit gate, $\mathcal{F}_j^{2q}$ is the average fidelity of the j-th two-qubit gate and the two summation operators run over all the quantum gates of the target quantum circuit. Defining as $\mathcal{F}^{1q}$ and $\mathcal{F}^{2q}$ the single and two-qubit native fidelities of the target hardware—i.e., the average fidelities of the native two-qubit gate and of a reference single-qubit gate (usually, the $R_X(\pi/2)$ gate) customarily reported in open-source calibration data repositories of real quantum computers—the reference article recommends to select K such that $\mathcal{F}^{1q} < K < \mathcal{F}^{2q}$. Accordingly, here the value of K is computed as

$$K = \frac{\mathcal{F}^{1q} + \mathcal{F}^{2q}}{2} \,. \tag{8}$$

For the target technologies of the proposed toolchain, the values of $\mathcal{F}^{1q}$ and $\mathcal{F}^{2q}$ are determined as discussed in the following:

– **Superconductors**: the fidelities are defined as

$$\begin{aligned} \mathcal{F}^{1q} &\triangleq \mathcal{F}(R_X(\pi/2)) \sim 1 - e_{1q} = 0.99926 \\ \mathcal{F}^{2q} &\triangleq \mathcal{F}(\mathrm{CX}) \sim 1 - e_{2q} = 0.97917 \,, \end{aligned} \tag{9}$$

where e_{1q} and e_{2q} are the average single ($\frac{\pi}{2}$-pulse) and two-qubit native gate error rates, respectively, retrieved from the calibration data of the *mock* backend `FakeToronto` [51] available in Qiskit Terra, which is substantially a simplified model for classical simulations of the twenty-seven-qubit IBMQ Toronto quantum computer. According to the previous formula, fidelity is assumed to be the complement of the error rate with respect to the unit.

– **Trapped ions**: the fidelities are

$$\begin{aligned} \mathcal{F}^{1q} &\triangleq \mathcal{F}(R_X(\pi/2)) = 0.99717 \\ \mathcal{F}^{2q} &\triangleq \mathcal{F}(R_{XX}) = 0.96960 \,, \end{aligned} \tag{10}$$

where $\mathcal{F}(R_X(\pi/2))$ and $\mathcal{F}(R_{XX})$ are the single and two-qubit native gate fidelities, respectively, and they are retrieved from [52].

– **NMR**: the fidelities are

$$\begin{aligned} \mathcal{F}^{1q} &\triangleq \mathcal{F}(R_X(\pi/2)) = 0.99895 \\ \mathcal{F}^{2q} &\triangleq \mathcal{F}(R_{ZZ}) = 0.97977 \,, \end{aligned} \tag{11}$$

where $\mathcal{F}(R_X(\pi/2))$ and $\mathcal{F}(R_{ZZ})$ are the single and two-qubit native gate fidelities, respectively, and they are computed exploiting the simulator proposed in [9]. More in detail, the physical parameters (J-couplings, resonance frequencies, decoherence and relaxation time constants [53–55]) of a four-qubit fully connected quantum computer based on a crotonic acid molecule are chosen as inputs to the compact model simulation infrastructure. Then, the fidelities resulting from the application of $R_X(\pi/2)$ pulses on each qubit and of R_{ZZ} gates on each couple of qubits are computed. Finally, the average fidelities are determined as follows:

$$\begin{aligned} \mathcal{F}(R_X(\pi/2)) &= \frac{1}{4}\sum_{i=0}^{3}\mathcal{F}_i(R_X(\pi/2)) \\ \mathcal{F}(R_{ZZ}) &= \frac{1}{6}\sum_{i=0}^{2}\sum_{j=i+1}^{3}\mathcal{F}_{i,j}(R_{ZZ})\,. \end{aligned} \tag{12}$$

Since the toolchain's output file exclusively contains the native gates of the target technology, it is reasonable to assume, as a first-order approximation, that all two-qubit gates show the same average fidelity. Therefore,

$$\mathcal{F}_i^{2q} = \mathcal{F}^{2q}\,,\ \forall i\,. \tag{13}$$

Conversely, since the execution time of single-qubit gates depends on the rotation angle, larger angles will lead to longer gate executions and, therefore, to higher error rates and lower fidelities. Hence, it is not acceptable to use the same single-qubit average fidelity for all single-qubit gates. It can be shown (see Appendix A) that there exists an approximated relation between the gate fidelity of an arbitrary single-qubit gate ($\mathcal{F}^{1q}(R(\theta))$) describing a rotation of an angle θ and the average fidelity of a $R_X(\pi/2)$ pulse ($\mathcal{F}(R_X(\pi/2))$):

$$\mathcal{F}^{1q}(R(\theta)) \sim 1 - \frac{2\,|\theta|}{\pi}\left(1 - \mathcal{F}(R_X(\pi/2))\right)\,. \tag{14}$$

- The **elapsed computation time (T)** for completing the whole compilation, in seconds.

As for the choice of quantum circuits to be tested, it was preferred to use general, different-sized quantum circuits. Most of them are available on two **GitHub** repositories:

1. **QASMBench** [56, 57], an existing QASM Benchmark Suite.
2. A repository of testing circuits belonging to **Prof. Dr. R. Wille's IIC Group from the Johannes Kepler University of Linz** [58]. These quantum circuits implement classical Boolean functions, such as addition.

The OpenQASM 2.0 descriptions of **SHOR** and **EDGE_DETECT**, reported in Tables 2, 3 and 4 and associated with Shor's algorithm and horizontal edge detection

Table 2 Subset of the benchmarks results for the **NMR technology** case with **average precision** toolchain parameters set

Circuit name	Qiskit Lv.1	Qiskit Lv.3	t|ket⟩	t|ket⟩ Max Opt.	Toolchain
ADDER_SMALL	**1Q**: 36 **1QNZ**: 30	**1Q**: 36 **1QNZ**: 30	**1Q**: 39 **1QNZ**: 4	**1Q**: 37 **1QNZ**: 4	**1Q**: 33 **1QNZ**: 18
(4 qubits,	**MQ**: 10 **L**: 42	**MQ**: 10 **L**: 42	**MQ**: 10 **L**: 6	**MQ**: 10 **L**: 6	**MQ**: 10 **L**: 18
1Q: 17,	**D**: 17 **C**: 0.430	**D**: 17 **C**: 0.430	**D**: 24 **C**: 0.467	**D**: 22 **C**: 0.446	**D**: 20 **C**: 0.436
MQ: 10)	**T**: 0.079 s	**T**: 0.092 s	**T**: 0.003 s	**T**: 0.029 s	**T**: 0.568 s
HS4	**1Q**: 30 **1QNZ**: 24	**1Q**: 24 **1QNZ**: 12	**1Q**: 32 **1QNZ**: 8	**1Q**: 16 **1QNZ**: 4	**1Q**: 18 **1QNZ**: 12
(4 qubits,	**MQ**: 16 **L**: 26	**MQ**: 4 **L**: 12	**MQ**: 4 **L**: 20	**MQ**: 2 **L**: 8	**MQ**: 4 **L**: 12
1Q: 24,	**D**: 21 **C**: 0.579	**D**: 9 **C**: 0.191	**D**: 16 **C**: 0.274	**D**: 8 **C**: 0.135	**D**: 8 **C**: 0.180
MQ: 4)	**T**: 0.084 s	**T**: 0.074 s	**T**: 0.003 s	**T**: 0.011 s	**T**: 0.489 s
DNN	**1Q**: 752 **1QNZ**: 440	**1Q**: 272 **1QNZ**: 136	**1Q**: 1172 **1QNZ**: 312	**1Q**: 432 **1QNZ**: 96	**1Q**: 840 **1QNZ**: 460
(8 qubits,	**MQ**: 192 **L**: 622	**MQ**: 64 **L**: 115	**MQ**: 192 **L**: 846	**MQ**: 64 **L**: 140	**MQ**: 192 **L**: 583
1Q: 2864,	**D**: 155 **C**: 6.236	**D**: 55 **C**: 2.016	**D**: 243 **C**: 7.414	**D**: 96 **C**: 2.482	**D**: 162 **C**: 6.269
MQ: 192)	**T**: 0.805 s	**T**: 0.979 s	**T**: 0.200 s	**T**: 0.341 s	**T**: 10.251 s
QPE	**1Q**: 80 **1QNZ**: 61	**1Q**: 79 **1QNZ**: 68	**1Q**: 181 **1QNZ**: 19	**1Q**: 174 **1QNZ**: 19	**1Q**: 137 **1QNZ**: 54
(8 qubits,	**MQ**: 43 **L**: 47	**MQ**: 43 **L**: 47	**MQ**: 43 **L**: 30	**MQ**: 43 **L**: 30	**MQ**: 43 **L**: 31
1Q: 79,	**D**: 90 **C**: 1.891	**D**: 79 **C**: 1.773	**D**: 132 **C**: 2.322	**D**: 131 **C**: 2.312	**D**: 124 **C**: 2.238
MQ: 43)	**T**: 0.132 s	**T**: 0.251 s	**T**: 0.011 s	**T**: 0.106 s	**T**: 1236.983 s
VQE_UCCSD	**1Q**: 11637 **1QNZ**: 10405	**1Q**: 10840 **1QNZ**: 9304	**1Q**: 15009 **1QNZ**: 1275	**1Q**: 13071 **1QNZ**: 1538	**1Q**: 9288 **1QNZ**: 6464
(8 qubits,	**MQ**: 5488 **L**: 14740	**MQ**: 4807 **L**: 13071	**MQ**: 5284 **L**: 5958	**MQ**: 4521 **L**: 1900	**MQ**: 5284 **L**: 7572
1Q: 5320,	**D**: 15597 **C**: 294.491	**D**: 13560 **C**: 257.030	**D**: 13226 **C**: 255.745	**D**: 10675 **C**: 208.584	**D**: 12774 **C**: 252.591
MQ: 5488)	**T**: 10.644 s	**T**: 49.287 s	**T**: 0.715 s	**T**: 36.859 s	**T**: 1230.643 s

Table 2 continued

Circuit name	Qiskit Lv.1	Qiskit Lv.3	t|ket⟩	t|ket⟩ Max Opt.	Toolchain
ISING	**1Q**: 267 **1QNZ**: 144	**1Q**: 293 **1QNZ**: 168	**1Q**: 400 **1QNZ**: 50	**1Q**: 400 **1QNZ**: 50	**1Q**: 369 **1QNZ**: 145
(10 qubits,	**MQ**: 90 **L**: 131	**MQ**: 90 **L**: 177	**MQ**: 90 **L**: 230	**MQ**: 90 **L**: 230	**MQ**: 90 **L**: 142
1Q: 390,	**D**: 57 **C**: 2.586	**D**: 61 **C**: 2.678	**D**: 80 **C**: 2.937	**D**: 80 **C**: 2.937	**D**: 82 **C**: 2.865
MQ: 90)	**T**: 0.256 s	**T**: 0.979 s	**T**: 0.034 s	**T**: 0.239 s	**T**: 1.275 s
SHOR	**1Q**: 25393 **1QNZ**: 17716	**1Q**: 22571 **1QNZ**: 17206	**1Q**: 52000 **1QNZ**: 4606	**1Q**: 52006 **1QNZ**: 4622	**1Q**: 47911 **1QNZ**: 16664
(12 qubits,	**MQ**: 14858 **L**: 13115	**MQ**: 14348 **L**: 12478	**MQ**: 14348 **L**: 4638	**MQ**: 14333 **L**: 4667	**MQ**: 14348 **L**: 11394
1Q: 20756,	**D**: 28623 **C**: 623.617	**D**: 28107 **C**: 607.005	**D**: 38336 **C**: 708.189	**D**: 38351 **C**: 708.074	**D**: 48547 **C**: 824.512
MQ: 14858)	**T**: 27.704 s	**T**: 88.706 s	**T**: 7.544 s	**T**: 42409.995 s	**T**: 15639.917 s
EDGE_DETECT	**1Q**: 37371 **1QNZ**: 10985	**1Q**: 17793 **1QNZ**: 10981	**1Q**: 81615 **1QNZ**: 16431	**1Q**: 57174 **1QNZ**: 11668	**1Q**: 44313 **1QNZ**: 16493
(12 qubits,	**MQ**: 16096 **L**: 840	**MQ**: 10875 **L**: 842	**MQ**: 16096 **L**: 18940	**MQ**: 11340 **L**: 14826	**MQ**: 16096 **L**: 1133
1Q: 48593,	**D**: 51666 **C**: 882.514	**D**: 27235 **C**: 514.474	**D**: 64413 **C**: 1037.894	**D**: 43501 **C**: 712.677	**D**: 57955 **C**: 950.095
MQ: 16096)	**T**: 29.199 s	**T**: 117.292 s	**T**: 8.469 s	**T**: 418.473 s	**T**: 36021.936 s
SYS6	**1Q**: 628 **1QNZ**: 490	**1Q**: 298 **1QNZ**: 183	**1Q**: 334 **1QNZ**: 22	**1Q**: 316 **1QNZ**: 22	**1Q**: 314 **1QNZ**: 162
(16 qubits,	**MQ**: 392 **L**: 567	**MQ**: 98 **L**: 200	**MQ**: 98 **L**: 136	**MQ**: 93 **L**: 136	**MQ**: 98 **L**: 143
1Q: 117,	**D**: 467 **C**: 13.603	**D**: 144 **C**: 3.753	**D**: 135 **C**: 3.590	**D**: 131 **C**: 3.445	**D**: 131 **C**: 3.554
MQ: 98)	**T**: 0.674 s	**T**: 1.855 s	**T**: 0.015 s	**T**: 0.195 s	**T**: 1.419 s
ADDER_LARGE	**1Q**: 339 **1QNZ**: 265	**1Q**: 393 **1QNZ**: 237	**1Q**: 469 **1QNZ**: 40	**1Q**: 461 **1QNZ**: 57	**1Q**: 413 **1QNZ**: 165
(18 qubits	**MQ**: 232 **L**: 272	**MQ**: 130 **L**: 303	**MQ**: 130 **L**: 48	**MQ**: 122 **L**: 68	**MQ**: 122 **L**: 119
1Q: 216,	**D**: 303 **C**: 8.268	**D**: 232 **C**: 5.457	**D**: 229 **C**: 5.157	**D**: 229 **C**: 5.014	**D**: 243 **C**: 5.218
MQ: 130)	**T**: 0.403 s	**T**: 2.072 s	**T**: 0.021 s	**T**: 0.245 s	**T**: 1.976 s

Table 2 continued

Circuit name	Qiskit Lv.1	Qiskit Lv.3	t|ket⟩	t|ket⟩ Max Opt.	Toolchain
URF5	**1Q**: 68287 **1QNZ**: 55730	**1Q**: 68196 **1QNZ**: 55671	**1Q**: 77023 **1QNZ**: 4370	**1Q**: 80312 **1QNZ**: 7172	**1Q**: 70123 **1QNZ**: 35390
(16 qubits,	**MQ**: 23764 **L**: 72257	**MQ**: 23726 **L**: 72168	**MQ**: 23742 **L**: 26140	**MQ**: 22919 **L**: 31086	**MQ**: 23681 **L**: 31545
1Q: 37505, **MQ**: 23764)	**D**: 47646 **C**: 1071.276 **T**: 62.493 s	**D**: 47608 **C**: 1069.999 **T**: 179.636 s	**D**: 49982 **C**: 1047.428 **T**: 5.420 s	**D**: 50973 **C**: 1046.404 **T**: 1290.521 s	**D**: 45765 **C**: 1006.666 **T**: 41711.981 s
SYM10	**1Q**: 180512 **1QNZ**: 140390	**1Q**: 84184 **1QNZ**: 50139	**1Q**: 93252 **1QNZ**: 4469	**1Q**: 96188 **1QNZ**: 6031	**1Q**: 88855 **1QNZ**: 44444
(16 qubits,	**MQ**: 112336 **L**: 164173	**MQ**: 28084 **L**: 56800	**MQ**: 28084 **L**: 30822	**MQ**: 28043 **L**: 33626	**MQ**: 28084 **L**: 38749
1Q: 52247, **MQ**: 28084)	**D**: 201745 **C**: 4626.449 **T**: 207.592 s	**D**: 59806 **C**: 1273.391 **T**: 455.926 s	**D**: 61266 **C**: 1261.809 **T**: 7.133 s	**D**: 62279 **C**: 1274.753 **T**: 1997.039 s	**D**: 55515 **C**: 1208.515 **T**: 58567.032 s

In this table and in the following ones, **1Q** represents the *number of single-qubit gates*, **1QNZ** represents the *number of non-R_Z single-qubit gates*, **MQ** represents the *number of multi-qubit gates*, **L** represents the *weighted latency for single-qubit gates* and **T** represents the *elapsed compilation time in seconds*. The number of gates in the original circuit, along with its related number of qubits, is reported alongside the reference circuit's name

Table 3 Subset of the benchmarks results for the **trapped ions technology** case with average precision toolchain parameters set

Circuit name	Qiskit Lv.1	Qiskit Lv.3	t|ket⟩	t|ket⟩ Max Opt.	Toolchain
ADDER_SMALL	**1Q**: 42 **1QNZ**: 35	**1Q**: 38 **1QNZ**: 31	**1Q**: 59 **1QNZ**: 44	**1Q**: 57 **1QNZ**: 44	**1Q**: 41 **1QNZ**: 32
(4 qubits,	**MQ**: 10 **L**: 39	**MQ**: 10 **L**: 37	**MQ**: 10 **L**: 226	**MQ**: 10 **L**: 226	**MQ**: 10 **L**: 35
1Q: 17,	**D**: 21 **C**: 0.770	**D**: 20 **C**: 0.747	**D**: 29 **C**: 1.440	**D**: 27 **C**: 1.407	**D**: 21 **C**: 0.758
MQ: 10)	**T**: 0.081 s	**T**: 0.112 s	**T**: 0.003 s	**T**: 0.030 s	**T**: 0.595 s
HS4	**1Q**: 14 **1QNZ**: 8	**1Q**: 20 **1QNZ**: 10	**1Q**: 40 **1QNZ**: 24	**1Q**: 20 **1QNZ**: 12	**1Q**: 16 **1QNZ**: 16
(4 qubits,	**MQ**: 4 **L**: 8	**MQ**: 4 **L**: 14	**MQ**: 4 **L**: 108	**MQ**: 2 **L**: 52	**MQ**: 4 **L**: 20
1Q: 24,	**D**: 7 **C**: 0.263	**D**: 9 **C**: 0.314	**D**: 18 **C**: 0.733	**D**: 9 **C**: 0.361	**D**: 8 **C**: 0.314
MQ: 4)	**T**: 0.086 s	**T**: 0.067 s	**T**: 0.003 s	**T**: 0.012 s	**T**: 0.538 s
DNN	**1Q**: 600 **1QNZ**: 380	**1Q**: 307 **1QNZ**: 133	**1Q**: 1556 **1QNZ**: 1080	**1Q**: 560 **1QNZ**: 352	**1Q**: 660 **1QNZ**: 336
(8 qubits,	**MQ**: 192 **L**: 414	**MQ**: 64 **L**: 158	**MQ**: 192 **L**: 5070	**MQ**: 64 **L**: 1548	**MQ**: 192 **L**: 373
1Q: 2864,	**D**: 151 **C**: 9.630	**D**: 57 **C**: 3.378	**D**: 279 **C**: 25.083	**D**: 107 **C**: 8.190	**D**: 163 **C**: 9.714
MQ: 192)	**T**: 0.758 s	**T**: 0.963 s	**T**: 0.210 s	**T**: 0.347 s	**T**: 12.148 s
QPE	**1Q**: 150 **1QNZ**: 100	**1Q**: 112 **1QNZ**: 62	**1Q**: 267 **1QNZ**: 191	**1Q**: 260 **1QNZ**: 191	**1Q**: 150 **1QNZ**: 78
(8 qubits,	**MQ**: 43 **L**: 106	**MQ**: 43 **L**: 96	**MQ**: 43 **L**: 976	**MQ**: 43 **L**: 976	**MQ**: 43 **L**: 100
1Q: 79,	**D**: 115 **C**: 3.555	**D**: 97 **C**: 3.227	**D**: 169 **C**: 6.947	**D**: 167 **C**: 6.913	**D**: 129 **C**: 3.772
MQ: 43)	**T**: 0.167 s	**T**: 0.367 s	**T**: 0.011 s	**T**: 0.105 s	**T**: 0.819 s
VQE_UCCSD	**1Q**: 17123 **1QNZ**: 15269	**1Q**: 12106 **1QNZ**: 9247	**1Q**: 25577 **1QNZ**: 22411	**1Q**: 22113 **1QNZ**: 19622	**1Q**: 11892 **1QNZ**: 10021
(8 qubits,	**MQ**: 5488 **L**: 15305	**MQ**: 4807 **L**: 12879	**MQ**: 5284 **L**: 122206	**MQ**: 4521 **L**: 101362	**MQ**: 5284 **L**: 13576
1Q: 5320,	**D**: 16275 **C**: 485.479	**D**: 11166 **C**: 372.011	**D**: 18472 **C**: 821.769	**D**: 15307 **C**: 685.613	**D**: 11646 **C**: 396.760
MQ: 5488)	**T**: 13.549 s	**T**: 76.751 s	**T**: 0.698 s	**T**: 36.442 s	**T**: 918.463 s

Table 3 continued

Circuit name	Qiskit Lv.1	Qiskit Lv.3	t|ket⟩	t|ket⟩ Max Opt.	Toolchain
ISING	**1Q**: 346 **1QNZ**: 193	**1Q**: 326 **1QNZ**: 176	**1Q**: 580 **1QNZ**: 410	**1Q**: 580 **1QNZ**: 410	**1Q**: 315 **1QNZ**: 134
(10 qubits,	**MQ**: 90 **L**: 189	**MQ**: 90 **L**: 208	**MQ**: 90 **L**: 2210	**MQ**: 90 **L**: 2210	**MQ**: 90 **L**: 144
1Q: 390)	**D**: 66 **C**: 4.418	**D**: 66 **C**: 4.474	**D**: 93 **C**: 10.651	**D**: 93 **C**: 10.651	**D**: 68 **C**: 4.327
MQ: 90	**T**: 0.313 s	**T**: 0.998 s	**T**: 0.034 s	**T**: 0.240 s	**T**: 1.206 s
SHOR	**1Q**: 46641 **1QNZ**: 30518	**1Q**: 36665 **1QNZ**: 19765	**1Q**: 80696 **1QNZ**: 61998	**1Q**: 80672 **1QNZ**: 61954	**1Q**: 38228 **1QNZ**: 14933
(12 qubits,	**MQ**: 14858 **L**: 29755	**MQ**: 14348 **L**: 29782	**MQ**: 14348 **L**: 320294	**MQ**: 14333 **L**: 319993	**MQ**: 14348 **L**: 14345
1Q: 20756)	**D**: 41138 **C**: 1232	**D**: 37451 **C**: 1153.046	**D**: 52644 **C**: 2240.043	**D**: 52654 **C**: 2238.886	**D**: 35026 **C**: 1070.450
MQ: 14858	**T**: 39.195 s	**T**: 112.683 s	**T**: 7.974 s	**T**: 49422.683 s	**T**: 9152.169 s
EDGE_DET:ECT:	**1Q**: 49685 **1QNZ**: 28385	**1Q**: 33146 **1QNZ**: 11846	**1Q**: 113807 **1QNZ**: 80815	**1Q**: 79854 **1QNZ**: 57028	**1Q**: 31971 **1QNZ**: 12687
(12 qubits,	**MQ**: 16096 **L**: 25986	**MQ**: 16096 **L**: 24409	**MQ**: 16096 **L**: 373052	**MQ**: 11340 **L**: 264306	**MQ**: 16096 **L**: 29216
1Q: 48593,	**D**: 48619 **C**: 1385.174	**D**: 46874 **C**: 1355.180	**D**: 79769 **C**: 2898.957	**D**: 54192 **C**: 2012.985	**D**: 45665 **C**: 1347.558
MQ: 16096)	**T**: 44.188 s	**T**: 118.950 s	**T**: 9.925 s	**T**: 562.087 s	**T**: 19055.206 s
SYS6	**1Q**: 378 **1QNZ**: 297	**1Q**: 351 **1QNZ**: 270	**1Q**: 530 **1QNZ**: 414	**1Q**: 502 **1QNZ**: 394	**1Q**: 350 **1QNZ**: 246
(16 qubits,	**MQ**: 98 **L**: 311	**MQ**: 98 **L**: 311	**MQ**: 98 **L**: 2292	**MQ**: 93 **L**: 2182	**MQ**: 98 **L**: 292
1Q: 117,	**D**: 160 **C**: 6.586	**D**: 155 **C**: 6.503	**D**: 190 **C**: 12.758	**D**: 188 **C**: 12.256	**D**: 138 **C**: 6.164
MQ: 98)	**T**: 0.292 s	**T**: 0.989 s	**T**: 0.016 s	**T**: 0.195 s	**T**: 1.149 s
ADDER_LARGE	**1Q**: 440 **1QNZ**: 327	**1Q**: 361 **1QNZ**: 249	**1Q**: 729 **1QNZ**: 560	**1Q**: 705 **1QNZ**: 545	**1Q**: 340 **1QNZ**: 206
(18 qubits	**MQ**: 130 **L**: 329	**MQ**: 130 **L**: 317	**MQ**: 130 **L**: 2908	**MQ**: 122 **L**: 2752	**MQ**: 122 **L**: 257
1Q: 216,	**D**: 257 **C**: 9.252	**D**: 236 **C**: 8.867	**D**: 328 **C**: 17.817	**D**: 327 **C**: 17.107	**D**: 175 **C**: 7.426
MQ: 130)	**T**: 0.360 s	**T**: 1.161 s	**T**: 0.037 s	**T**: 0.358 s	**T**: 1.423 s

Table 3 continued

Circuit name	Qiskit Lv.1	Qiskit Lv.3	t|ket⟩	t|ket⟩ Max Opt.	Toolchain
URF5	**1Q**: 84681 **1QNZ**: 67934	**1Q**: 74557 **1QNZ**: 57810	**1Q**: 124507 **1QNZ**: 99338	**1Q**: 126150 **1QNZ**: 98848	**1Q**: 70461 **1QNZ**: 49335
(16 qubits,	**MQ**: 23764 **L**: 71017	**MQ**: 23742 **L**: 70176	**MQ**: 23742 **L**: 548464	**MQ**: 22919 **L**: 535303.5	**MQ**: 23681 **L**: 65096
1Q: 37505, **MQ**: 23764)	**D**: 55503 **C**: 1864.850 **T**: 60.785 s	**D**: 52386 **C**: 1809.690 **T**: 195.306 s	**D**: 72147 **C**: 3508.877 **T**: 5.564 s	**D**: 71830 **C**: 3440.435 **T**: 1149.855 s	**D**: 146006 **C**: 1686.575 **T**: 26300.822 s
SYM10	**1Q**: 102759 **1QNZ**: 80250	**1Q**: 93170 **1QNZ**: 70661	**1Q**: 149420 **1QNZ**: 97627	**1Q**: 152274 **1QNZ**: 118203	**1Q**: 88877 **1QNZ**: 59575
(16 qubits,	**MQ**: 28084 **L**: 84560	**MQ**: 28084 **L**: 84560	**MQ**: 28084 **L**: 648670	**MQ**: 28043 **L**: 650572	**MQ**: 28084 **L**: 77906
1Q: 52247, **MQ**: 28084)	**D**: 67259 **C**: 2233.573 **T**: 70.981 s	**D**: 65238 **C**: 2199.84 **T**: 230.274 s	**D**: 88996 **C**: 4211.564 **T**: 7.398 s	**D**: 89730 **C**: 4227.963 **T**: 1987.166 s	**D**: 55513 **C**: 2018.121 **T**: 38863.062 s

Table 4 Subset of the benchmarks results for the **superconducting technology** case with average precision toolchain parameters set

Circuit name	Qiskit Lv.1	Qiskit Lv.3	t|ket⟩	t|ket⟩ Max Opt.	Toolchain
ADDER_SMALL	**1Q**: 10 **1QNZ**: 4	**1Q**: 11 **1QNZ**: 4	**1Q**: 11 **1QNZ**: 4	**1Q**: 10 **1QNZ**: 4	**1Q**: 9 **1QNZ**: 4
(4 qubits,	**MQ**: 10 **L**: 6	**MQ**: 10 **L**: 6	**MQ**: 10 **L**: 8	**MQ**: 10 **L**: 8	**MQ**: 10 **L**: 6
1Q: 17,	**D**: 12 **C**: 0.345	**D**: 12 **C**: 0.345	**D**: 13 **C**: 0.357	**D**: 12 **C**: 0.346	**D**: 12 **C**: 0.345
MQ: 10)	**T**: 0.054 s	**T**: 0.043 s	**T**: 0.003 s	**T**: 0.028 s	**T**: 0.561 s
HS4	**1Q**: 8 **1QNZ**: 8	**1Q**: 8 **1QNZ**: 8	**1Q**: 8 **1QNZ**: 8	**1Q**: 4 **1QNZ**: 4	**1Q**: 6 **1QNZ**: 6
(4 qubits,	**MQ**: 4 **L**: 8	**MQ**: 4 **L**: 8	**MQ**: 4 **L**: 16	**MQ**: 2 **L**: 8	**MQ**: 4 **L**: 8
1Q: 24,	**D**: 6 **C**: 0.155	**D**: 6 **C**: 0.155	**D**: 6 **C**: 0.161	**D**: 4 **C**: 0.091	**D**: 5 **C**: 0.144
MQ: 4)	**T**: 0.057 s	**T**: 0.052 s	**T**: 0.002 s	**T**: 0.012 s	**T**: 0.510 s
DNN	**1Q**: 328 **1QNZ**: 312	**1Q**: 136 **1QNZ**: 136	**1Q**: 360 **1QNZ**: 312	**1Q**: 128 **1QNZ**: 96	**1Q**: 328 **1QNZ**: 328
(8 qubits,	**MQ**: 192 **L**: 464	**MQ**: 64 **L**: 268	**MQ**: 192 **L**: 624	**MQ**: 64 **L**: 192	**MQ**: 192 **L**: 528
1Q: 2864,	**D**: 98 **C**: 5.444	**D**: 34 **C**: 1.912	**D**: 98 **C**: 5.562	**D**: 34 **C**: 1.857	**D**: 98 **C**: 5.491
MQ: 192)	**T**: 0.765 s	**T**: 1.036 s	**T**: 0.202 s	**T**: 0.390 s	**T**: 7.722 s
QPE	**1Q**: 71 **1QNZ**: 19	**1Q**: 59 **1QNZ**: 19	**1Q**: 57 **1QNZ**: 19	**1Q**: 56 **1QNZ**: 19	**1Q**: 68 **1QNZ**: 16
(8 qubits,	**MQ**: 43 **L**: 20	**MQ**: 43 **L**: 20	**MQ**: 43 **L**: 38	**MQ**: 43 **L**: 38	**MQ**: 43 **L**: 19
1Q: 79,	**D**: 83 **C**: 1.819	**D**: 72 **C**: 1.700	**D**: 73 **C**: 1.720	**D**: 72 **C**: 1.713	**D**: 82 **C**: 1.808
MQ: 43)	**T**: 0.091 s	**T**: 0.246 s	**T**: 0.010 s	**T**: 0.127 s	**T**: 0.649 s
VQE_UCCSD	**1Q**: 1891 **1QNZ**: 1275	**1Q**: 2972 **1QNZ**: 2370	**1Q**: 1891 **1QNZ**: 1275	**1Q**: 2177 **1QNZ**: 1538	**1Q**: 1891 **1QNZ**: 1275
(8 qubits,	**MQ**: 5284 **L**: 1324	**MQ**: 4807 **L**: 2805	**MQ**: 5284 **L**: 2550	**MQ**: 4521 **L**: 3076	**MQ**: 5284 **L**: 2539
1Q: 5320,	**D**: 6452 **C**: 182.128	**D**: 6424 **C**: 172.872	**D**: 6452 **C**: 183.029	**D**: 5881 **C**: 161.166	**D**: 6452 **C**: 183.02
MQ: 5488)	**T**: 3.254 s	**T**: 24.053 s	**T**: 0.542 s	**T**: 36.650 s	**T**: 280.067 s

Table 4 continued

Circuit name	Qiskit Lv.1	Qiskit Lv.3	t|ket⟩	t|ket⟩ Max Opt.	Toolchain
ISING	**1Q**: 145 **1QNZ**: 50	**1Q**: 137 **1QNZ**: 50	**1Q**: 120 **1QNZ**: 50	**1Q**: 120 **1QNZ**: 50	**1Q**: 145 **1QNZ**: 60
(10 qubits,	**MQ**: 90 **L**: 90	**MQ**: 90 **L**: 90	**MQ**: 90 **L**: 100	**MQ**: 90 **L**: 100	**MQ**: 90 **L**: 110
1Q: 390,	**D**: 42 **C**: 2.416	**D**: 42 **C**: 2.416	**D**: 42 **C**: 2.423	**D**: 42 **C**: 2.423	**D**: 42 **C**: 2.430
MQ: 90)	**T**: 0.178 s	**T**: 2.416 s	**T**: 0.030 s	**T**: 0.245 s	**T**: 0.791 s
SHOR	**1Q**: 18446 **1QNZ**: 4606	**1Q**: 16646 **1QNZ**: 5626	**1Q**: 14092 **1QNZ**: 4606	**1Q**: 14101 **1QNZ**: 4622	**1Q**: 18445 **1QNZ**: 4606
(12 qubits,	**MQ**: 14348 **L**: 4606	**MQ**: 14348 **L**: 6646	**MQ**: 14348 **L**: 9212	**MQ**: 14333 **L**: 9244	**MQ**: 14348 **L**: 7162
1Q: 20756,	**D**: 25812 **C**: 585.174	**D**: 26067 **C**: 589.436	**D**: 25551 **C**: 585.729	**D**: 25545 **C**: 585.372	**D**: 25812 **C**: 587.0518
MQ: 14858)	**T**: 10.197 s	**T**: 62.782 s	**T**: 585.729 s	**T**: 49274.067 s	**T**: 1008.194 s
EDGE_DET:ECT:	**1Q**: 16559 **1QNZ**: 16431	**1Q**: 11677 **1QNZ**: 11538	**1Q**: 16563 **1QNZ**: 16431	**1Q**: 11798 **1QNZ**: 11668	**1Q**: 16404 **1QNZ**: 10327
(12 qubits,	**MQ**: 16096 **L**: 27010	**MQ**: 10939 **L**: 21766	**MQ**: 16096 **L**: 32862	**MQ**: 11340 **L**: 23336	**MQ**: 16096 **L**: 18661
1Q: 48593,	**D**: 31742 **C**: 702.699	**D**: 21040 **C**: 474.297	**D**: 31745 **C**: 707.03	**D**: 21623 **C**: 490.210	**D**: 31587 **C**: 694.888
MQ: 16096)	**T**: 24.350 s	**T**: 97.250 s	**T**: 8.308 s	**T**: 543.127 s	**T**: 9388.558 s
SYS6	**1Q**: 94 **1QNZ**: 22	**1Q**: 94 **1QNZ**: 22	**1Q**: 94 **1QNZ**: 22	**1Q**: 86 **1QNZ**: 22	**1Q**: 88 **1QNZ**: 22
(16 qubits,	**MQ**: 98 **L**: 22	**MQ**: 98 **L**: 22	**MQ**: 98 **L**: 44	**MQ**: 93 **L**: 44	**MQ**: 98 **L**: 22
1Q: 117,	**D**: 73 **C**: 2.870	**D**: 73 **C**: 2.870	**D**: 73 **C**: 2.886	**D**: 71 **C**: 2.759	**D**: 72 **C**: 2.859
MQ: 98)	**T**: 0.093 s	**T**: 0.397 s	**T**: 0.012 s	**T**: 0.181 s	**T**: 0.608 s
ADDER_LARGE	**1Q**: 136 **1QNZ**: 40	**1Q**: 129 **1QNZ**: 40	**1Q**: 129 **1QNZ**: 40	**1Q**: 143 **1QNZ**: 57	**1Q**: 134 **1QNZ**: 39
(18 qubits	**MQ**: 130 **L**: 48	**MQ**: 130 **L**: 48	**MQ**: 130 **L**: 80	**MQ**: 122 **L**: 114	**MQ**: 122 **L**: 52
1Q: 216,	**D**: 149 **C**: 4.387	**D**: 149 **C**: 4.387	**D**: 150 **C**: 4.421	**D**: 155 **C**: 4.442	**D**: 134 **C**: 4.058
MQ: 130)	**T**: 0.118 s	**T**: 0.551 s	**T**: 0.017 s	**T**: 0.243 s	**T**: 0.649 s

Table 4 continued

Circuit name	Qiskit Lv.1	Qiskit Lv.3	t|ket⟩	t|ket⟩ Max Opt.	Toolchain
URF5	**1Q**: 20770 **1QNZ**: 4351	**1Q**: 20768 **1QNZ**: 4351	**1Q**: 20799 **1QNZ**: 4370	**1Q**: 20745 **1QNZ**: 7172	**1Q**: 20758 **1QNZ**: 4329
(16 qubits,	**MQ**: 23742 **L**: 4616	**MQ**: 23726 **L**: 4616	**MQ**: 23742 **L**: 8740	**MQ**: 22919 **L**: 14344	**MQ**: 23681 **L**: 4611
1Q: 37505, **MQ**: 23764)	**D**: 25671 **C**: 781.380 **T**: 13.094 s	**D**: 25660 **C**: 780.924 **T**: 96.074 s	**D**: 25587 **C**: 783.500 **T**: 4.069 s	**D**: 1216.437 **C**: 768.027 **T**: 1216.43 s	**D**: 25637 **C**: 779.724 **T**: 696.211 s
SYM10	**1Q**: 28086 **1QNZ**: 4462	**1Q**: 28086 **1QNZ**: 4462	**1Q**: 28146 **1QNZ**: 4469	**1Q**: 28111 **1QNZ**: 6031	**1Q**: 28085 **1QNZ**: 4455
(16 qubits,	**MQ**: 28084 **L**: 4544	**MQ**: 28084 **L**: 4544	**MQ**: 28084 **L**: 8938	**MQ**: 28043 **L**: 12062	**MQ**: 28084 **L**: 4536
1Q: 52247, **MQ**: 28084)	**D**: 31641 **C**: 937.432 **T**: 16.901 s	**D**: 31641 **C**: 937.431 **T**: 109.210 s	**D**: 31655 **C**: 940.811 **T**: 5.603 s	**D**: 31783 **C**: 943.630 **T**: 2068.116 s	**D**: 31640 **C**: 937.415 **T**: 512.660 s

in image processing, were obtained using the Qiskit transpiler on the original circuits available in [41]. The set of circuits chosen for the benchmark was composed with the idea of involving small-to-large-scaled circuits. Only the latest version of each chosen circuit in the repositories was taken in consideration. The circuits were mostly left untouched: the only actions performed on them were adding a measurement on each involved qubit line at the end and, in a few circuits taken form QASMBench's repository, to manually decompose custom user-defined gates, which the toolchain does not currently recognize nor support. Each circuital compilation executed by the toolchain was performed with three different sets of parameters: **low precision** ($\text{thr}_1 = 10^{-4}, \text{thr}_2 = 10^{-6}$), **average precision** ($\text{thr}_1 = 10^{-8}, \text{thr}_2 = 10^{-10}$) and **high precision** ($\text{thr}_1 = 10^{-10}, \text{thr}_2 = 10^{-12}$), in which thr_1 is the grade of approximation of π and thr_2 is the threshold of rotation value under which a gate is considered a null operation. All the compilations were performed on a single-process **Intel(R) Xeon(R) Gold 6134 CPU @ 3.20GHz opta-core, Model 85** [59] with a memory of 10296102+ KiB.

4.4 Comparison with the state of the art

The benchmarks were performed using **Qiskit version 0.28**, with **Qiskit Aer Libraries version 0.8.2** and **Qiskit Terra Libraries version 0.18.0** (the latest releases at the time of writing). The benchmarks were performed using **pytket-qiskit version 0.16.1** too. When using both compilers and the toolchain, each quantum architecture was considered as **perfectly fully connected**, not taking thus into account the mapping capabilities in the benchmarks.

It is important to clarify that Qiskit compilation for NMR employed a slightly different basis set of quantum gates. In fact, the set R_X, R_Y, R_Z, R_{ZZ} is not supported by the employed version of this library, so the R_{ZZ} was replaced by the CZ. Reminding that $\text{CZ} = R_{ZZ}\left(\frac{\pi}{2}\right)\left[R_Z\left(-\frac{\pi}{2}\right) \otimes R_Z\left(-\frac{\pi}{2}\right)\right]$ [10], this compilation strategy did not affect the calculation of the total number of two-qubit gates and of non-R_Z single-qubit gates, but it could slightly modify the circuit depth, which is generally expected a bit lower, considering that a single CZ requires two single-qubit gates and one two-qubit gate. In any case, this different basis set was employed to compare the NMR toolchain capabilities with a potentially better basis set in terms of circuit depth and cost.

The general trend noted in the benchmarks is that **Qiskit** features a versatile management of **single-qubit gates**, being able to optimize them well using the sets of gates of multiple target technologies. **t|ket⟩**, on the other hand, proved to be more **unpredictable** when dealing with single-qubit gates, because it alternates very good optimizations to completely suboptimal handles. At the same time, t|ket⟩ proved to be the best in the **reduction of two-qubit gates**, as it was able to reduce their number substantially by accepting some tradeoffs on the single-qubit gates number.

4.5 Obtained results

A subset of the obtained results is reported in Tables 2, 3, 4. It is important to precise that in the calculation of the normalized circuit latency (L), all non-integer results were

rounded-up to stay in a worse-case scenario. All the toolchain-produced circuits proved to be **functionally equivalent** to the reference input circuits. Among the evaluated KL deviations, none exceeded the value of 10^{-5}, thus proving that, even when multiple output states were expected, the obtained results were remarkably similar to the ones of the original circuits. Moreover, the toolchain proved capable of optimizing efficiently the **single-qubit gates** in most circuits. Some potentially powerful tools, like the Euler-combo optimization, proved to be very situational, while other processes, such as the IBM's U gates merging scheme, proved to be competitive even when compared to IBM's own software platorm's results. The toolchain also proved capable of **handling two-qubit gates quite well**, introducing optimizations capable of occasionally outperforming Qiskit and, most importantly, of **competing with t|ket⟩** in some cases, whose strong point is actually the proficiency in reducing the number of multi-qubit gates involved in the circuit. This went against the expectation of the template-based approach being unsuited to tackle efficiently the management of clusters of CX gates, as the templates that involve them are not numerous and as other methods, such as the heuristic-based ones, seem theoretically more prone to detect advantageous circuit remodelings. As expected, the number of single-qubit gates *drammatically increases* all over the chart in some technologies, since each decomposition of two-qubit gates introduces several single-qubit gates in the circuit and can bring a lot of newly inserted gates in large-sized circuits.

The specific results obtained for each technology are commented in the following:

- *NMR technology* Of all the supported technologies, the generated circuits with the NMR as target technology proved to be the ones in which the toolchain's optimization **is least competitive**. As shown in Table 2, when compared to **Qiskit**, the toolchain performs quite well, in particular with larger circuits, since it can involve less *physical* quantum gates (involving both single and two qubits). Moreover, in most of the examined circuits, the advantage in the management of single-qubit-gate implies a lower weighted latency. On the other hand, **t|ket⟩ obtains better performance than the other compilers**. In fact, considering both optimization levels, it always finds the solution with the lowest number of **two-qubit gates** and in most of the cases it achieves the best results in managing **single-qubit gates**, in terms of both the total number of non-R_Z gates and circuit latency. However, for what concerns the circuit cost, t|ket⟩ is sometimes *beat* by Qiskit and by the toolchain, especially with medium-large circuits, because of a generally higher circuit depth. In any case, even though the circuit depth and cost are sometimes the lowest, the benchmarks showed that the the currently implemented toolchain is not more performing than other state-of-the-art compilers.
- *Trapped ions technology* The benchmarks showed that the trapped ions technology is the **most successfully** managed by the toolchain. As shown in Table 3, even though the total number of two-qubit gates is not the lowest among all compilation strategies, the toolchain often achieves a consistent reduction of the number of all single-qubit gates and of those non-R_Z, specially in medium-large circuits (lower half of table). Moreover, the obtained equivalent circuits are generally characterized by low circuit depths, which permit to obtain circuit costs which are the lowest or competitive with those obtained with other compilation strategies.

Qiskit is capable of holding its ground quite well, while the same cannot be said for **t|ket⟩**, which features a less efficient implementation, apart from small quantum circuits; even if it manages well two-qubit gates, it is usually incapable of severely reducing single-qubit gates. Moreover, it also employs non-R_Z gates, with a very high associated weighted latency. The results obtained by the toolchain are particularly satisfying in the context of this technology, even more considering that chains of trapped ions are intrinsically fully connected. This means that the toolchain, even in this prototype form, is already capable of efficiently handling most scenarios involving quantum circuits implemented with trapped ions.

- *Superconducting technology* The toolchain shows quite good performance with this target technology. As shown in Table 4, **t|ket⟩** is usually able to minimize the total number of two-qubit gates, while managing at the same time the single-qubit U gates. Moreover, it is possible to ascertain that the maximum optimization level does not always imply better results, since in some cases the lower optimization level can generate an equivalent circuit with fewer two-qubit gates or circuit latency. **Qiskit** handles IBM's gate set really well in terms of single-qubit gates number, but it is often *beat* by t|ket⟩ in CX optimization. The toolchain achieves in most of the cases competitive weighted latencies with respect to both the other compilers. The same thing can be substantially said for circuit depth, thus implying quite good costs, to the point that in two cases (**ADDER_LARGE** and **SYM10**) this quantity is the lowest. Generally speaking, the advantages over the results of the other compilers are not very strong, but the overall performance can be considered satisfactory, according to the current state of the presented work. However, it has to be noted that, in this specific technology, the critical layout mapping phase was not taken in consideration, and that the smart management of SWAP gates to adapt circuits to target devices is one of Qiskit's and t|ket⟩'s strong point: perhaps, with such feature implemented the toolchain would prove less ideal than Qiskit, t|ket⟩ or both.

As for the differences between the toolchain compilations using different sets of parameters, in the **NMR** and **superconducting** technologies circuits obtained with the **high precision** set proved to be slightly more optimized, although the differences between different sets of parameters were minimal. In the **trapped ions** technology case, a trade-off emerged: in most circuits, the **higher precision** was used for the parameters, the **stronger the optimization of single-qubit gates** was, but also the **worse the weighted latency** became. The computation time also proved to be a critical parameter. In the superconducting technology case, the compilation time of the toolchain was non-negligible in the case of large-sized circuits, but not excessively long (even lower than the corresponding one for t|ket⟩ with highest optimization level). Combining the running times of all the steps, the toolchain takes usually longer than both Qiskit and t|ket⟩, but in small to medium-size circuits this difference is negligible and is still acceptable in large circuits. In the NMR and trapped ions technologies cases, the compilation time of the toolchain was in general significant and absolutely crippling when dealing with large-sized circuits. This is probably due to the employment of the Eulercombo mechanism in a circuit whose number of gates dramatically increased after the decomposition of CX and CZ gates.

When considering the overall running time of the toolchain, it is clear that the template-based approach can be fast enough for small to average circuits, but it is also extremely slower than other heuristic methods when dealing with very large circuits. This is mostly due to the issue of **Gimbal Locks** [60] in the Eulercombo process, which drastically slow the compilation, especially when using highly precise approximations of π.

4.5.1 Summary of the toolchain's most notable highlights

A small summary of benchmark cases in which the toolchain's performance stood out in terms of **number of non-RZ quantum gates**, **weighted latency**, **circuit depth** and **circuit cost**—when compared to the other state-of-the-art compilers—is here reported. Compilation times and scalability issues were not considered in the following highlights.

- In **NMR** technology, the toolchain achieves the lowest circuit cost in **URF5** and **SYM10** cases, because of its lowest circuit depth. However, it is not possible to say that it generally performs better than the other two compilers, especially when compared with t|ket⟩, because of its good management of quantum gates with this technology. On the other hand, when compared solely to Qiskit, the toolchain produced circuits with a total number of single-qubit gates and weighted latency equal or lesser than Qiskit's ones in all the tested cases, except for **DNN**, **ISING** and **EDGE_DETECT**.
- In **Trapped ions** technology, the proposed toolchain yielded the overall best results, in terms of both single-qubit latency and cost, for **ISING**, **ADDER_LARGE**, **SYM10**, **SHOR**, **SYS6** and **URF5**. In the first two cases, the compiled circuits have the lowest number of non-R_z single-qubit gates, two-qubit gates and the lowest circuit depth, so the overall performance can be considered the best. In the other cases, the cost function is minimized thanks to lower circuit depth, number of non-R_z single-qubit gates and weighted latency, compensating a *worse* management of two-qubit gates. **ADDER_SMALL** has the lowest single-qubit weighted latency, but the circuit cost is slightly higher than the one obtained with Qiskit with maximum optimization level, even though the total number of two-qubit gates is equal in both cases. The reason of this behaviour can be justified by the fact that the circuit depth is slightly higher (21 instead of 20). Finally, **EDGE_DETECT** has the lowest circuit cost, given by the lowest circuit depth (45665, with the second lowest equal to 46874), such as the total number of single-qubit gates (31971). On the other hand, the solution provided by the toolchain has a higher non-R_Z gates count (12687, instead of 11846) and single-qubit-gate weighted latency (29216, instead of 24409) with respect to Qiskit with maximum optimization level. In general, the obtained result can be considered competitive with respect to those provided by the other compilers.
- In **superconducting** technology, the toolchain achieved the minimum cost, among all the considered compilation strategies, with **ADDER_LARGE** and **SYM10** circuits, thanks to its lowest circuit depth. In the **ADDER_SMALL** case, it performed definitely on par with all the other compilers, especially with Qiskit. Finally, **QPE**,

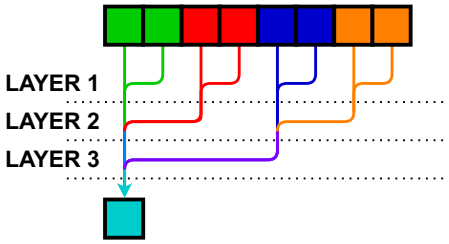


Fig. 13 Binary-tree reduction

EDGE_DETECT and **SYS6** can be mentioned, as they are characterized by the best obtained performance in terms of weighted latency, but not capable of achieving the minimum cost because of the higher circuit depth and the number of two-qubit gates equal or greater than the ones of Qiskit and t|ket⟩ compilers with maximum optimization level.

5 Conclusions and future perspectives

The obtained results show that the toolchain and its core philosophy can be considered competitive in the state of the art for the compilation of quantum circuits targeting specific technologies and that the designed optimization process can introduce some fine-grain technology-dependent optimizations that allow the toolchain to compete with well-established compilers in some cases, especially when dealing with trapped ions. In general, the toolchain is capable of steadily reducing the number of single-qubit gates and prioritizing an abundant use of advantageous and virtual R_Z gates, thus obtaining in some circumstances circuits with low depth and cost. However, it is important to remind that the toolchain evantual advantages would be compensated by compilation times usually very long. The largest circuits used in the benchmarks are probably too large in scale for current quantum computers to be able to handle them and actually use them for computation and use tens of thousands of gates, that once decomposed increase dramatically. In conclusion, the obtained results show a limit of some of the proposed optimizations, with compilations that require up to several hours to be completed. As reported through this paper, the toolchain is, however, still a prototype. Its structure could allow it to support even more quantum technologies, enhancing its already good versatility.

Regarding future improvements, a first step would be to extend the currently supported selection of quantum technologies. For instance, spin-based technologies are characterized by sets of native single and two-qubit gates similar to the ones of the NMR technology. Therefore, the routines and functions developed to optimize the compilation for NMR backends are expected to be easily adaptable to other spin-based technologies, such as quantum dots in semiconductors [61]. Moreover, the native gates of technologies already available could be updated and increased. This is, in particular, the case of superconducting qubits, where R_{ZX}, controlled phase (`cu1`, according to OpenQASM 2.0 syntax) and iSWAP could be taken into account in compilation, thus permitting on one hand to operate on a gate level closer to the physical one, and on

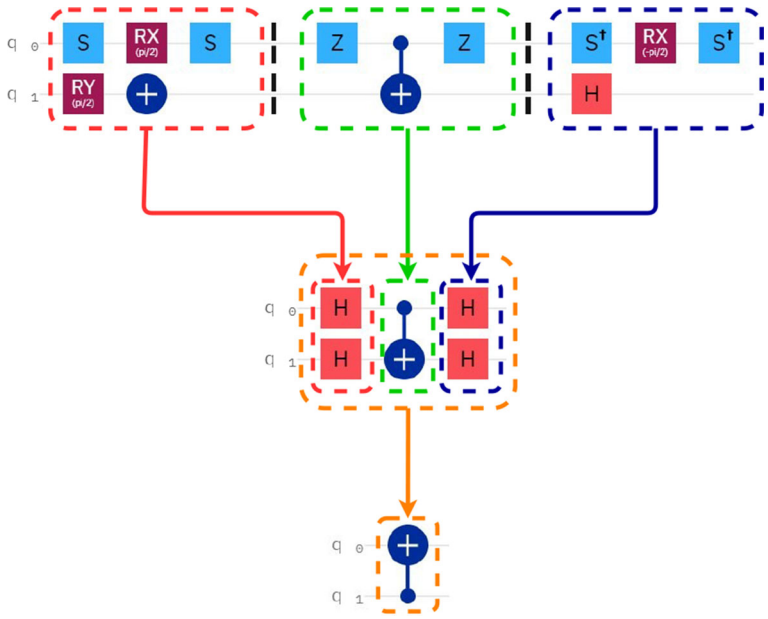


Fig. 14 Template-based optimization of a circuit with three sub-circuits

the other to employ the toolchain with backends of providers different from IBM's ones. It is clear that considering new native two-qubit gates for superconducting qubits could also imply the identification of new templates, associated with these gates, to be integrated in the toolchain.

The most relevant part currently missing in the toolchain is the capability of handling the whole layout synthesis (placement and routing) process for non-fully connected technologies, such as the superconducting one, and optimizing the strengths of couplings for fully connected technologies, such as the NMR one. This feature would require implementing a tool capable of adapting the compilation to a given device's layout and logically mapping each qubit line with a smart insertion of SWAP gates. In this regard, it shall be remarked that the compilation toolchain discussed in this article belongs to a broader project currently under development. The latter, in fact, shall include both a compilation part (proposed in this manuscript)—dedicated to compiling the quantum input circuit by reducing the quantum gates as much as possible, regardless of the error rate of each gate—and a placement and routing part (which at the time of writing is still missing)—with the aim of implementing a hardware-driven noise-and-fidelity-aware mapping of logical qubits to physical qubits, minimizing the execution time and maximizing the fidelity.

Moreover, in order to improve the compilation, other steps could be implemented in the toolchain, existing steps may also be modified to accommodate new features and currently implemented functions could be moved in the workflow to improve the overall process.

Last but not least, an effort in optimizing the overall software development could be done to specifically reduce compilation time. Apart from employing a more time-efficient programming language, e.g., C or C++, parallelization can be exploited. As already introduced in Sect. 4.2, a first approach of its can be accomplished by enacting a mechanism of sub-circuit optimization: the input quantum circuit can be divided into N sub-circuits, where N is chosen according to some criteria (depth of the algorithm, number of qubits, heuristics resulting from previous results). Then, the N sub-circuits are compiled in parallel, exploiting a reasonable number of threads and processes, according to the available hardware platform. Eventually, the optimized sub-circuits can be recombined together to force the toolchain to perform a last optimization of the merged circuit, according to a **tree-like organization** , such as the binary one reported in Fig. 13. A *naïf* example is shown in Fig. 14, where a quantum circuit is firstly divided into three initially independent sub-circuits—on each of which a template-based optimization can be executed in parallel—then these circuits are merged together and an additional optimization can be finally done, which permits to obtain a final circuit with only a two-qubit gate. If template-based approach worked properly on the sub-circuits, it is expected that the final circuit will show a reduced number of quantum gates. Therefore, the last optimization shall require less time, and the overall process can lead to a significant speed-up in terms of compilation time.

In conclusion, the proposed toolchain prototype produced some interesting results and showed a valuable potential for improvement, especially dealing with trapped ion technology. Even though the current status of the compiler clearly shows some weaknesses performance-wise, its core methodology, i.e., the technology-specific local optimization, proved to be an interesting foundation on which to build on, and is hoped to be an inspiring first step in a structured development process that could aim to achieve truly competitive results on the stage of the state of the art and a high grade of versatility in terms of supported technologies. With some spot-on optimizations to build on what this core has established and a certain degree of skill synergy to polish it up on different fields of expertise, the legacy of the toolchain is believed to evolve well in a tool that can truly push to the limit the potential of this compilation method in the context of quantum computation research.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s11128-022-03649-9.

Funding Open access funding provided by Politecnico di Torino within the CRUI-CARE Agreement.

Data availability The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

A Derivation of single-qubit-gate fidelity equation

Equation 14 was derived starting from the time evolution of the fidelity of a single-qubit stat—initially pure and then increasingly more mixed—under the effects of relaxation and decoherence, two non-ideality phenomena common to all the analyzed quantum computing technologies that usually provide quite significant contributions to the unwanted evolution of the qubit state. The proof of the fidelity equation is reported in the following for two particular cases.

First of all, it is important to remind that the fidelity between a pure state, described by a state vector $|\psi\rangle$, and a mixed state, described by a density matrix ρ, can be written as [62]:

$$\mathcal{F} \triangleq \sqrt{\langle\psi|\,\rho\,|\psi\rangle}\,. \tag{15}$$

This is a real scalar number between 0 and 1 and its maximum value is achieved when $\rho = |\psi\rangle\,\langle\psi|$. Some references [63] define fidelity without the square root:

$$\mathcal{F} \triangleq \langle\psi|\,\rho\,|\psi\rangle\ . \tag{16}$$

It is clear that, independently on the notation, the product $\langle\psi|\,\rho\,|\psi\rangle$ must be always computed. Before reporting the examples, it is important to further clarify that the computed fidelity is between the vector associated with the initial state of qubit ($t = 0$) and the density matrix associated with the qubit state after a time interval $\Delta t = t - 0 = t$. In the following, only the variable t (or τ, when referring to quantum gates duration) will be reported, but it must be interpreted as a time interval with respect to $t = 0$.

The first considered case is associated with the state

$$|\psi\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, \tag{17}$$

whose density matrix affected by relaxation and decoherence phenomena is [62]:

$$\rho = \begin{bmatrix} 1 - \frac{1}{2}e^{-\frac{t}{T_1}} & \frac{1}{2}e^{-\frac{t}{T_2}} \\ \frac{1}{2}e^{-\frac{t}{T_2}} & \frac{1}{2}e^{-\frac{t}{T_1}} \end{bmatrix}, \tag{18}$$

where t is the time variable and T_1 and T_2 are characteristic time constants of relaxation and decoherence respectively. For $t = 0$, the density matrix corresponds to $|\psi\rangle\,\langle\psi|$ and the fidelity is equal to 1. It is possible to prove that:

$$\begin{aligned} \rho\,|\psi\rangle &= \frac{1}{2\sqrt{2}} \begin{bmatrix} 2 - e^{-\frac{t}{T_1}} + e^{-\frac{t}{T_2}} \\ e^{-\frac{t}{T_1}} + e^{-\frac{t}{T_2}} \end{bmatrix}, \\ \langle\psi|\,\rho\,|\psi\rangle &= \frac{1 + e^{-\frac{t}{T_2}}}{2}\,. \end{aligned} \tag{19}$$

In this particular case, relaxation does not affect fidelity. Assuming to be in a *low-timescale* regime, where $t \ll T_2$, the exponential can be approximated with a first-order Maclaurin-Taylor expansion:

$$\begin{aligned} \langle\psi|\,\rho\,|\psi\rangle &= \frac{1+e^{-\frac{t}{T_2}}}{2} \approx \frac{1+1-\frac{t}{T_2}}{2} = 1-\frac{t}{2T_2}\,, \\ \sqrt{\langle\psi|\,\rho\,|\psi\rangle} &\approx \sqrt{1-\frac{t}{2T_2}} \approx 1-\frac{t}{4T_2}\,. \end{aligned} \tag{20}$$

Equation 20 puts in evidence that, independently on the definition, $\mathcal{F}$ can be described by a descending straight line $1-kt$, with real $k>0$ depending on $\frac{1}{T_2}$.

The fidelity of a single-qubit gate was properly interpreted in terms of the evolution of a qubit state under the previously mentioned phenomena, on a time interval $\Delta t = t-0$ corresponding to the duration of the gate τ, assumed to be always much lower than T_1 and T_2, as typical in experimental quantum computing. This approach neglects the effect of the quantum gate itself on the qubit state, but permits to employ the same *simple low-timescale* model in all the examined compilation cases, without any arrangement depending on the technology. If the duration of a $\frac{\pi}{2}$-gate is assumed to be known and equal to $\tau_{\frac{\pi}{2}}$ independently on the rotation axis, the fidelity can be written as:

$$\mathcal{F}(R_X(\pi/2)) \approx 1-k\tau_{\frac{\pi}{2}}\,, \tag{21}$$

where $\mathcal{F}(R_X(\pi/2))$ is the fidelity of a $R_x\left(\frac{\pi}{2}\right)$, often reported in the experimental characterization data of a quantum computer (explicitly on implicitly, in terms of its dual error rate). Reminding that the duration of a generic single-qubit gate with rotation θ is proportional to $\tau_{\frac{\pi}{2}}$:

$$\tau_\theta = \frac{2|\theta|}{\pi}\tau_{\frac{\pi}{2}}\,, \tag{22}$$

its corresponding fidelity can be written as:

$$\begin{aligned} \mathcal{F}^{1q}(R(\theta)) &= 1-k\tau_\theta = 1-k\frac{2|\theta|}{\pi}\tau_{\frac{\pi}{2}} \\ &= 1-k\frac{2|\theta|}{\pi}\tau_{\frac{\pi}{2}} + k\tau_{\frac{\pi}{2}} - k\tau_{\frac{\pi}{2}} \\ &= (1-k\tau_{\frac{\pi}{2}}) + \left(1-\frac{2|\theta|}{\pi}\right)k\tau_{\frac{\pi}{2}} \\ &= \mathcal{F}(R_X(\pi/2)) \\ &\quad + \left(1-\frac{2|\theta|}{\pi}\right)(1-\mathcal{F}(R_X(\pi/2)))\,. \end{aligned} \tag{23}$$

An analogous result can be obtained for a more general case

$$
\begin{aligned}
|\psi\rangle &= \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}, \\
\rho &= \begin{bmatrix} 1 + (c_0^2 - 1)e^{-\frac{t}{T_1}} & c_0\sqrt{1-c_0^2}\,e^{-\frac{t}{T_2}} \\ c_0\sqrt{1-c_0^2}\,e^{-\frac{t}{T_2}} & (1-c_0^2)e^{-\frac{t}{T_1}} \end{bmatrix},
\end{aligned}
\tag{24}
$$

where c_0 and $c_1 = \sqrt{1-c_0^2}$ are two **real probability amplitudes**. It is possible to prove that:

$$
\begin{aligned}
\rho\,|\psi\rangle &= \begin{bmatrix} c_0\left[1 + (1-c_0^2)(e^{-\frac{t}{T_2}} - e^{-\frac{t}{T_1}})\right] \\ \sqrt{1-c_0^2}\left[c_0^2 e^{-\frac{t}{T_2}} + (1-c_0^2)e^{-\frac{t}{T_1}})\right] \end{bmatrix}, \\
\langle\psi|\,\rho\,|\psi\rangle &= c_0^2 + (1 + 2c_0^4 - 3c_0^2)e^{-\frac{t}{T_1}} \\
&\quad + 2c_0^2(1-c_0^2)e^{-\frac{t}{T_2}}\,.
\end{aligned}
\tag{25}
$$

In the *low-timescale* regime ($t \ll T_1, T_2$), the first-order Maclaurin-Taylor approximation can be employed, thus:

$$
\begin{aligned}
\langle\psi|\,\rho\,|\psi\rangle &\approx 1 - \left[\frac{1 + 2c_0^4 - 3c_0^2}{T_1} + \frac{2c_0^2(1-c_0^2)}{T_2}\right]t\,, \\
\sqrt{\langle\psi|\,\rho\,|\psi\rangle} &\approx 1 - \frac{1}{2}\left[\frac{1 + 2c_0^4 - 3c_0^2}{T_1} + \frac{2c_0^2(1-c_0^2)}{T_2}\right]t\,.
\end{aligned}
\tag{26}
$$

It is possible to prove that, for $c_0 = c_1 = \frac{1}{\sqrt{2}}$, the results correspond to those in Equation 20. The linear descending function $\mathcal{F} = 1 - kt$ is valid also in this case, so — apart from a generic k depending on both $\frac{1}{T_2}$ and $\frac{1}{T_1}$ and on the initial qubit measurement probabilities c_0^2 and $\sqrt{1-c_0^2}$—Eqs. 21, 22 and 23, obtained for the particular case $|\psi\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix}$, can be employed straightforwardly.

In conclusion, it is important to clarify that the same trend is expected to be also valid for the single-qubit state described by generic complex probability amplitudes. In that case, k will depend on $\frac{1}{T_1}$ and $\frac{1}{T_2}$ (as in the already examined cases) and on the magnitude square values of the complex probability amplitudes $|c_0|^2$ and $|c_1|^2 = 1 - |c_0|^2$.

References

1. Soeken, M., Häner, T., Roetteler, M.: Programming quantum computers using design automation. (2018). arXiv:1803.01022

2. Preskill, J.: Lecture notes on quantum information and computation. http://theory.caltech.edu/~preskill/ph229/. Accessed 3-January-2022
3. Loredo, R.: Learn Quantum computing with python and IBM Quantum Experience: A hands-on introduction to quantum computing and writing your own quantum programs with Python. Packt Publishing Ltd, 2020. https://www.packtpub.com/product/learn-quantum-computing-with-python-and-ibm-quantum-experience/9781838981006
4. Jang, W., Terashi, K., Saito, M., Bauer, C. W., Nachman, B., Iiyama, Y., Kishimoto, T., Okubo, R., Sawada, R., Tanaka, J.: Quantum gate pattern recognition and circuit optimization for scientific applications. In: EPJ Web Conf., vol. 251, p. 03023, 2021. https://doi.org/10.1051/epjconf/202125103023
5. Munoz-Coreas, E.: Resource efficient design of quantum circuits for cryptanalysis and scientific computing applications. PhD thesis, University of Kentucky, Electrical and Computer Engineering (2020). https://doi.org/10.13023/etd.2020.365
6. Hietala, K., Rand, R., Hung, S.-H., Wu, X., Hicks, M.: A verified optimizer for quantum circuits. Proc. ACM Program. Lang., vol. 5 (2021)
7. Fox, M.: Quantum Optics: An Introduction. Oxford Master Series in Atomic, Optical, and Laser Physics. Oxford Univ. Press, Oxford (2006). https://cds.cern.ch/record/1001868
8. Cirillo, G.A., Turvani, G., Graziano, M.: A quantum computation model for molecular nanomagnets. IEEE Trans. Nanotechnol. **18**, 1027–1039 (2019). https://doi.org/10.1109/TNANO.2019.2939910
9. Simoni, M., Cirillo, G.A., Turvani, G., Graziano, M., Zamboni, M.: Towards compact modeling of noisy quantum computers: a molecular-spin-qubit case of study. J. Emerg. Technol. Comput. Syst. **18**, 1–26 (2021)
10. Cirillo, G. A., Turvani, G., Simoni, M., Graziano, M.: Advances in molecular quantum computing: from technological modeling to circuit design. In: 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 132–137 (2020)
11. Parhi, K.K.: VLSI Digital Signal Processing Systems: Design and Implementation. Wiley, Hoboken (1999)
12. "Exercise 4 IBM quantum challenge 2020. https://github.com/qiskit-community/may4_challenge_exercises/tree/master/ex04. Accessed 3-January-2022
13. IBM Quantum Experience, quantum composer. https://quantum-computing.ibm.com/composer. Accessed 25-October-2021
14. Vandersypen, L.M.K., Chuang, I.L.: NMR techniques for quantum control and computation. Rev. Mod. Phys. **76**, 1037–1069 (2005). https://doi.org/10.1103/RevModPhys.76.1037
15. Bruzewicz, C.D., Chiaverini, J., McConnell, R., Sage, J.M.: Trapped-ion quantum computing: progress and challenges. Appl. Phys. Rev. **6**, 021314 (2019). https://doi.org/10.1063/1.5088164
16. Huang, H.-L., Wu, D., Fan, D., Zhu, X.: Superconducting quantum computing: a review. Sci. China Inf. Sci. (2020). https://doi.org/10.1007/s11432-020-2881-9
17. Architecture and design automation for quantum computing. https://vast.cs.ucla.edu/projects/architecture-and-compilation-quantum-computing. Accessed 25-Oct-2021
18. Saeedi, M., Markov, I.L.: Synthesis and optimization of reversible circuits–a survey. ACM Comput. Surv. **45**, 1–34 (2013). https://doi.org/10.1145/2431211.2431220
19. Davis, M. G., Smith, E., Tudor, A., Sen, K., Siddiqi, I., Iancu, C.: Heuristics for quantum compiling with a continuous gate set (2019). arXiv:1912.02727
20. Sanders, Y.R., Berry, D.W., Costa, P.C., Tessler, L.W., Wiebe, N., Gidney, C., Neven, H., Babbush, R.: Compilation of fault-tolerant quantum heuristics for combinatorial optimization. PRX Quantum **1**, 020312 (2020). https://doi.org/10.1103/PRXQuantum.1.020312
21. Hogg, T.: Quantum search heuristics. Phys. Rev. A **61**, 052311 (2000). https://doi.org/10.1103/PhysRevA.61.052311
22. Biswal, L., Das, R., Bandyopadhyay, C., Chattopadhyay, A., Rahaman, H.: A template-based technique for efficient Clifford+T-based quantum circuit implementation. Microelectron. J. **81**, 58–68 (2018). https://doi.org/10.1016/j.mejo.2018.08.011
23. LaRose, R.: Overview and comparison of gate level quantum software platforms. Quantum **3**, 130 (2019)
24. ANIS, M. S., Abraham, H., Agarwal, AduOffei, R., Agliardi, G., Aharoni, M., Akhalwaya, I. Y., Aleksandrowicz, G., Alexander, T., Amy, M., Anagolum, S., Arbel, E., Asfaw, A., et al., Qiskit: an open-source framework for quantum computing. (2021). https://doi.org/10.5281/zenodo.2573505

25. Javadi-Abhari, A., Nation, P., Gambetta, J.: Qiskit - write once, target multiple architectures. [Online] https://www.ibm.com/blogs/research/2019/11/qiskit-for-multiple-architectures/. Accessed 25-October-2021
26. Wille, R., Van Meter, R., Naveh, Y.: IBM's Qiskit tool chain: working with and developing for real quantum computers. 2019 Design, Automation & Test in Europe Conference and Exhibition (DATE). 2019. https://doi.org/10.23919/DATE.2019.8715261
27. "t|ket⟩ - quantum software development platform." [Online] https://cambridgequantum.com/technology/, accessed 25-October-2021
28. Sivarajah, S., Dilkes, S., Cowtan, A., Simmons, W., Edgington, A., Duncan, R.: t|ket⟩: a retargetable compiler for NISQ devices. Quantum Sci. Technol. **6**, 014003 (2020). https://doi.org/10.1088/2058-9565/ab8e92
29. Cross, A. W., Bishop, L. S., Smolin, J. A., Gambetta, J. M.: Open quantum assembly language. (2017). arXiv:1707.03429
30. Cambridge quantum's tket is now open-sourced. https://cambridgequantum.com/cambridge-quantums-tket-is-now-open-sourced/. Accessed 25-October-2021
31. Qiskit's optimization module. https://qiskit.org/documentation/stable/0.28/apidoc/qiskit_optimization.html. Accessed 26-June-2022
32. IBM decision optimization CPLEX modeling for python. https://ibmdecisionoptimization.github.io/docplex-doc/. Accessed 26-June-2022
33. Sivarajah, S., Dilkes, S., Cowtan, A., Simmons, W., Edgington, A., Duncan, R.: t|ket⟩ : a retargetable compiler for NISQ devices (2020). https://doi.org/10.48550/arXiv.2003.10611
34. Tucci, R. R.: An introduction to Cartan's KAK decomposition for QC programmers (2005)
35. Itoko, T., Raymond, R., Imamichi, T., Matsuo, A.: Optimization of quantum circuit mapping using gate transformation and commutation. (2019). arXiv:1907.02686
36. McKay, D.C., Wood, C.J., Sheldon, S., Chow, J.M., Gambetta, J.M.: Efficient z gates for quantum computing. Phys. Rev. A **96**, 022330 (2017). https://doi.org/10.1103/PhysRevA.96.022330
37. Circuit library—template circuits. https://qiskit.org/documentation/apidoc/circuit_library.html. Accessed 26-June-2022
38. Shende, V.V., Markov, I.L.: On the CNOT-cost of TOFFOLI gates. Quantum Inf. Comput **9**(5), 461–486 (2009). https://doi.org/10.26421/QIC9.5-6-8
39. Barenco, A., Bennett, C.H., Cleve, R., DiVincenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J.A., Weinfurter, H.: Elementary gates for quantum computation. Phys. Rev. A **52**, 3457–3467 (1995). https://doi.org/10.1103/PhysRevA.52.3457
40. Garcia-Escartin, J. C., Chamorro-Posada, P.: Equivalent quantum circuits. (2011). arXiv:1110.2998
41. Asfaw, A., Corcoles, A., Bello, L., Ben-Haim, Y., Bozzo-Rey, M., Bravyi, S., Bronn, N., Capelluto, L., Vazquez, A. C., et al..: Learn quantum computation using qiskit. (2020). http://community.qiskit.org/textbook
42. Rynbach, A. V., Muhammad, A., Mehta, A. C., Hussmann, J., Kim, J.: A quantum performance simulator based on fidelity and fault-path counting (2012)
43. Ben-Ari, M.: A tutorial on Euler angles and quaternions. https://www.weizmann.ac.il/sci-tea/benari/sites/sci-tea.benari/files/uploads/softwareAndLearningMaterials/quaternion-tutorial-2-0-1.pdf. Accessed 25-Oct-2021
44. Zhang, X., Xiang, H., Xiang, T., Fu, L., Sang, J.: An efficient quantum circuits optimizing scheme compared with qiskit. (2018). arXiv:1807.01703
45. Krantz, P., Kjaergaard, M., Yan, F., Orlando, T.P., Gustavsson, S., Oliver, W.D.: A quantum engineer's guide to superconducting qubits. Appl. Phys. Rev. **6**(2), 021318 (2019). https://doi.org/10.1063/1.5089550
46. Burkard, G., Ladd, T. D., Nichol, J. M., Pan, A., Petta, J. R.: Semiconductor spin qubits (2021). arXiv:2112.08863
47. Abobeih, M.H., Wang, Y., Randall, J., Loenen, S.J.H., Bradley, C.E., Markham, M., Twitchen, D.J., Terhal, B.M., Taminiau, T.H.: Fault-tolerant operation of a logical qubit in a diamond quantum processor. Nature **606**, 884–889 (2022). https://doi.org/10.1038/s41586-022-04819-6
48. Kullback, S., Leibler, R.A.: On information and sufficiency. Ann. Math. Stat. **22**(1), 79–86 (1951). https://doi.org/10.1214/aoms/1177729694
49. Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., Saraiva, J.: Energy efficiency across programming languages: how do energy, time, and memory relate?. In: Proceedings of the 10th

ACM SIGPLAN International Conference on Software Language Engineering, pp. 256–267 (2017). https://doi.org/10.1145/3136014.3136031
50. Kharkov, Y., Ivanova, A., Mikhantiev, E., Kotelnikov, A.: Arline benchmarks: automated benchmarking platform for quantum compilers. (2022). https://doi.org/10.48550/ARXIV.2202.14025
51. FakeToronto backend calibration data. https://github.com/Qiskit/qiskit-terra/tree/main/qiskit/providers/fake_provider/backends/toronto. Accessed 02-July-2022
52. IonQ API Calibrations. https://quantumai.google/cirq/hardware/ionq/calibrations. Accessed 08-July-2022
53. National Institute of Advanced Industrial Science and Technology, Spectral database for organic compounds (2022). https://sdbs.db.aist.go.jp. Accessed 30-June-2022
54. Li, K., Li, Y., Han, M., Lu, S., Zhou, J., Ruan, D., Long, G., Wan, Y., Lu, D., Zeng, B., Laflamme, R.: Quantum spacetime on a quantum simulator. Commun. Phys. **2**, 122–128 (2019). https://doi.org/10.1038/s42005-019-0218-5
55. Wen, J., Kong, X., Wei, S., Wang, B., Xin, T., Long, G.: Experimental realization of quantum algorithms for a linear system inspired by adiabatic quantum computing. Phys. Rev. A **99**, 012320 (2019). https://doi.org/10.1103/PhysRevA.99.012320
56. Li, A., Stein, S., Krishnamoorthy, S., Ang, J.: QASMBench: A low-level QASM benchmark suite for nisq evaluation and simulation. (2021). arXiv:2005.13018
57. QASMBench circuits repository. https://github.com/uudown/QASMBench. Accessed 25-Oct-2021
58. JKU IIC circuits repository. https://github.com/iic-jku/ibm_qx_mapping/tree/master/examples. Accessed 25-Oct-2021
59. Intel Xeon Gold 6134 processor - product specification. [Online] https://ark.intel.com/content/www/us/en/ark/products/120493/intel-xeon-gold-6134-processor-24-75m-cache-3-20-ghz.html. Accessed 25-October-2021
60. Brezov, D. S., Mladenova, C. D., Mladenov, I. M.: New perspective on the gimbal lock problem. In: AIP Conference Proceedings, vol. 1570, pp. 367–374, American Institute of Physics, 2013. https://doi.org/10.1063/1.4854778
61. Burkard, G., Ladd, T. D., Nichol, J. M., Pan, A., Petta, J. R.: Semiconductor spin qubits (2021). arXiv:2112.08863v1
62. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press, Cambridge (2010). https://doi.org/10.1017/CBO9780511976667
63. Jozsa, R.: Fidelity for mixed quantum states. J. Mod. Opt. **41**(12), 2315–2323 (1994). https://doi.org/10.1080/09500349414552171
64. Maslov, D.: Basic circuit compilation techniques for an ion-trap quantum machine. New J. Phys. **19**, 023035 (2017). https://doi.org/10.1088/1367-2630/aa5e47