Received August 3, 2020; revised October 19, 2020; accepted January 19, 2021; date of publication January 22, 2021; date of current version March 18, 2021.

Digital Object Identifier 10.1109/TQE.2021.3053921

Compiler Design for Distributed Quantum Computing

DAVIDE FERRARI[1] (Graduate Student Member, IEEE), ANGELA SARA CACCIAPUOTI[2,3] (Senior Member, IEEE), MICHELE AMORETTI[1] (Senior Member, IEEE), AND MARCELLO CALEFFI[2,3] (Senior Member, IEEE)

[1]Department of Engineering and Architecture, University of Parma, 43124 Parma, Italy
[2]Future Communications Laboratory, Department of Electrical Engineering and Information Technology, University of Naples Federico II, 80125 Naples, Italy
[3]Laboratorio Nazionale di Comunicazioni Multimediali, National Inter-University Consortium for Telecommunications, 80126 Naples, Italy

Corresponding author: Davide Ferrari (davide.ferrari1@unipr.it)

The work of Angela Sara Cacciapuoti and Marcello Caleffi was supported in part by the project "Towards the Quantum Internet: A Multidisciplinary Effort," University of Naples Federico II, Italy, and by the Italian MoD PNRM research program "QuaSaR: Quantum" Safe netwoRk".

ABSTRACT In distributed quantum computing architectures, with the network and communications functionalities provided by the Quantum Internet, remote quantum processing units can communicate and cooperate for executing computational tasks that single, noisy, intermediate-scale quantum devices cannot handle by themselves. To this aim, distributed quantum computing requires a new generation of quantum compilers, for mapping any quantum algorithm to any distributed quantum computing architecture. With this perspective, in this article, we first discuss the main challenges arising with compiler design for distributed quantum computing. Then, we analytically derive an upper bound of the overhead induced by quantum compilation for distributed quantum computing. The derived bound accounts for the overhead induced by the underlying computing architecture *as well as* the additional overhead induced by the suboptimal quantum compiler—expressly designed in this article to achieve three key features, namely, *general-purpose*, *efficient*, and *effective*. Finally, we validate the analytical results, and we confirm the validity of the compiler design through an extensive performance analysis.

INDEX TERMS Distributed quantum computing, distributed quantum systems, quantum compiling, quantum Internet, quantum networks.

I. INTRODUCTION

Current quantum computers are commonly defined as *noisy intermediate-scale quantum (NISQ)* devices, being characterized by a few dozens of quantum bits (qubits) with nonuniform quality and highly constrained physical connectivity.

Hence, the growing demand for large-scale quantum computers is motivating research on distributed quantum computing architectures [1]–[3], and experimental efforts have demonstrated some of the building blocks for such a design [4]. Indeed, with the network and communication functionalities provided by the *Quantum Internet* [2], [3], [5]–[12], remote quantum processing units (QPUs) can communicate and cooperate—through the distributed computing paradigm as a *virtual quantum processor* with a number of qubits that scales linearly with the number of remote QPUs [13]—for executing computational tasks that each NISQ device cannot handle by itself.

As overviewed in recent literature such as [4] and [13], several challenges arise with the design of a distributed quantum computing architecture. In the following, we focus on the problem of designing a *quantum algorithm compiler* for distributed quantum compilation.

Compiling a quantum algorithm means translating a hardware-agnostic description of the algorithm—i.e., the quantum circuit[1]—into a functionally equivalent one that takes into account the physical constraints of the underlying computing architecture—i.e., the *compiled* quantum circuit. When it comes to distributed computing architectures, there are two main issues arising with the compiler design.

[1]See Section II for a proper introduction to quantum circuits, circuit compilation, and circuit depth.

First, a fundamental question arises with distributed computation: *at what price*? Indeed, distributed computation requires the different processors being able to communicate with each other for coordinating and data exchanging, and these tasks introduce an overhead that strongly depend on the particulars of the distributed computing architecture. For instance, the induced overhead becomes more severe as the connectivity between the QPUs shrinks or as the number of qubits stored at the QPUs decreases. Hence, from a compiling perspective, it is crucial to estimate the overhead effects onto the compiled quantum circuit, effects that are generally measured in terms of *depth* of the compiled circuit with respect to the depth of the original one.

Furthermore, compiling a quantum circuit is a very challenging task even for a single-processor architecture, being such a task an NP-complete problem [14]. Hence, optimal circuit compiling for distributed quantum architectures can be achieved only for very small circuit instances. Conversely, the compilation of medium to large circuits of practical value induces an additional overhead—whose severity depends on the suboptimality of the quantum compiler—that further increases the depth of the compiled circuit.

With this in mind, in this article, we analytically derive an upper bound of the overhead induced by quantum circuit compilation for distributed quantum computing:

1) by considering the overhead induced by the worst-case scenario for a distributed quantum computing architecture, namely, a scenario characterized by a) the lowest possible number of qubits at each QPU and b) the *poorest* connectivity among the QPUs;
2) by considering the additional overhead induced by a suboptimal quantum compiler.

Clearly, with reference to the last point, the additional overhead strongly depends on the particulars of the quantum compiler. To this aim, in this article, we design a quantum compiler with three key features:

1) *general-purpose*, namely, requiring no particular assumptions on the quantum circuits to be compiled;
2) *efficient*, namely, exhibiting a polynomial-time computational complexity so that it can successfully compile medium-to-large circuits of practical value;
3) *effective*, being the total circuit depth overhead induced by the quantum circuit compilation always upper-bounded by a factor that grows linearly with the number of logical qubits of the original quantum circuit.

The rest of this article is organized as follows. In Section II, we review some preliminaries about quantum circuits and quantum compilers. Then, in Section III, we detail the problem of circuit compilation for distributed quantum computing, discussing the challenges that arise with the compiler design and the relevant literature. These basics are crucial for understanding the compiler design as well as the analytical derivation of the overhead bound given in Section IV. Then, Section V presents the performance analysis for the proposed compiler design. In particular, we present the implementation of a compiler that is able to cope with the worst-case scenario for a distributed quantum computing architecture; we validate the compiling overhead upper bound; we illustrate experimental results regarding the compilation of several quantum circuits, with our compiler compared to a state-of-the-art solution. Finally, Section VI concludes this article.

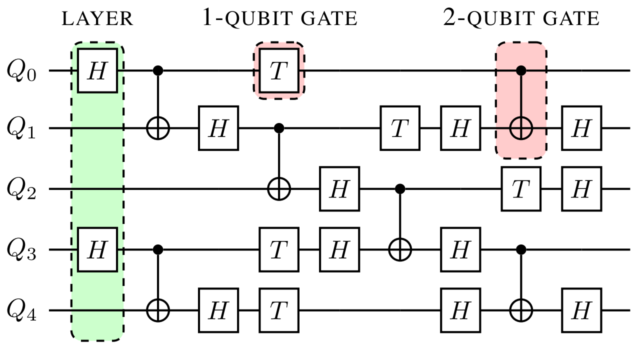


FIG. 1. **Example of a 5-qubit quantum circuit from [18], with each horizontal line representing the time evolution of the state of a single logical qubit.**

II. BACKGROUND

We refer the reader to [15] for an introduction to the conceptual and notation differences separating quantum computing from conventional computing, and to [16] for an in-depth treatise of the subject.

In this article, we consider QPUs that support the *quantum circuit model* [17], which is the most popular and developed model for quantum computation. A quantum circuit is a model of a quantum algorithm, where quantum operators are described as *quantum gates*. A quantum circuit is still a logical abstraction, not to be confused with its realization on an actual quantum hardware device. Hence, in the following, the abstract qubits subjected to quantum gates as specified by the quantum circuit are called *logical qubits* to distinguish them from the *physical qubits* embedded within a quantum processor.

Fig. 1 shows a simple quantum circuit, where each horizontal line represents the time evolution of the state of a single logical qubit, with time flowing from left to right, dictating the order of execution of the different gates. More specifically, gates affecting the same qubit must be executed sequentially, and this agrees with the intuition. Conversely, gates acting on different qubits can be performed simultaneously as long as the "ordering" arising from gates affecting multiple qubits is respected. This concept underlies the notion of *layer*, i.e., the set of gates that can be performed simultaneously on a disjoint set of qubits. The number of layers in a quantum circuit is denoted as *circuit depth*. As an example, the quantum circuit given in Fig. 1 is composed

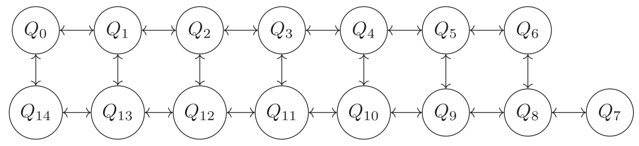


FIG. 2. Coupling map of the IBM Melbourne quantum processor [22]. The 15 physical qubits are represented by circles. The arrows denote the possibility to realize a two-qubit `CNOT` gate between the connected qubits, with the arrow pointing toward the target qubit. As an example, a `CNOT` between qubits Q_1 (control) and Q_0 (target) can be directly executed by the quantum processor, whereas a `CNOT` between qubits Q_2 and Q_0 cannot.

of nine layers, and hence, its depth is equal to 9. The number of gates within the circuit is denoted as *circuit size*.

A. QUANTUM COMPILATION

Given a quantum algorithm, there exist several equivalent quantum circuits modeling the same computation with a different arrangement or different ordering of gates.

Circuits with fewer gates—i.e., with lower *size*—may be preferred to reduce the circuit complexity. However, the execution time of the circuit—rather than its size—is generally considered the key factor to be optimized [19], [20]. The rationale is to keep the execution time of the quantum circuit within the coherence time of the underlying quantum hardware architecture [16], [21]. By oversimplifying, the execution time increases with the number of layers. Therefore, it is crucial to build—for a given quantum algorithm—a quantum circuit characterized by the lowest possible depth. However, two issues arise as a consequence of the quantum processor characteristics.

First, even if there exist an uncountable number of quantum logic gates, the set of gates that can be executed on a certain quantum processor can be limited, as a consequence of the constraints imposed by the underlying qubit technology [4]. In this case, any gate outside this *reduced set* must be obtained with a proper combination of the allowed gates through a process known as *gate synthesis*.

Furthermore, regardless of the underlying qubit technology, any quantum processor exhibits physical constraints—arising as a consequence of the noise and the physical space limitations—on the possible interactions between the different physical qubits. For example, `CNOT` gates cannot be applied to any physical qubit pair, but they are instead restricted to certain pairs, as shown in Fig. 2, with the *coupling map* of an IBM quantum processor.

From the above, it becomes clear that the execution of a quantum algorithm on a certain quantum processor requires that: 1) each logical qubit of the quantum circuit is mapped[2] onto a physical qubit of the quantum processor; and 2) each `CNOT` operation between nonadjacent (within the coupling map) physical qubits is mapped into a sequence of `CNOT` operations between adjacent physical qubits, as shown[3] in Fig. 3.

This process, known as *quantum compilation*, must be optimized so that the depth of the *compiled circuit*—i.e., the equivalent quantum circuit satisfying all the constrains imposed by the quantum processor—is minimized [25], [26], [28], [29].

III. COMPILERS FOR DISTRIBUTED QUANTUM COMPUTING

As highlighted in Section I, the demand for large-scale quantum computers is motivating research on distributed quantum computing architectures, where multiple small-scale quantum processors interact and cooperate through the Quantum Internet for solving challenging computational tasks. As a consequence, a new generation of quantum compilers is needed for mapping any quantum algorithm to any distributed quantum computing architecture.

Let us consider a toy model for distributed quantum computing, in which a generic quantum algorithm must be executed on two quantum processors interconnected by a quantum link, as shown in Fig. 4.

A. CHALLENGES

Several challenges arise with the design of a quantum compiler for mapping an arbitrary quantum circuit into a distributed quantum computing architecture, as discussed in the following.

1) DATA QUBITS VERSUS COMMUNICATION QUBITS

Similarly to classical distributed computing, a key requirement for distributed quantum computing is the possibility to perform *remote operations*, namely, operations between qubits stored at different processors. However, differently from the classical domain, quantum mechanics does not allow an unknown qubit to be copied or even simply read or measured in any way, without causing an irreversible loss of the quantum information stored within the qubit [15], [16].

Thankfully, *entanglement* provides an invaluable tool for implementing remote operations without violating quantum mechanics [13]. Entanglement is a property of two (or more, in case of *multipartite* entanglement) quantum particles that exist in a special type of superposition state, such that any action on a particle affects instantaneously the other particle as well. This sort of quantum correlation, with no counterpart in the classical world, holds even when the particles are far away from each other. For an in-depth discussion about entanglement from an information engineering point of view, we refer the reader to [21].

By exploiting the availability of a *Bell state*—that is a state of two maximally-enatngled qubits (EPR pairs, where EPR stands for Einstein, Podolsky, and Rosen)—shared between the two remote processors, it is possible to perform a remote

[2]Indeed, NISQ technology may require a logical qubit to be mapped onto several physical qubits to implement proper fault-tolerant techniques [24] Nevertheless, in the following, we assume a one-to-one mapping for the sake of clarity, without any loss of generality.

[3]With the *state transfer* strategy based on `SWAP`s usually preferred over the *ancilla* strategy [25]–[27].

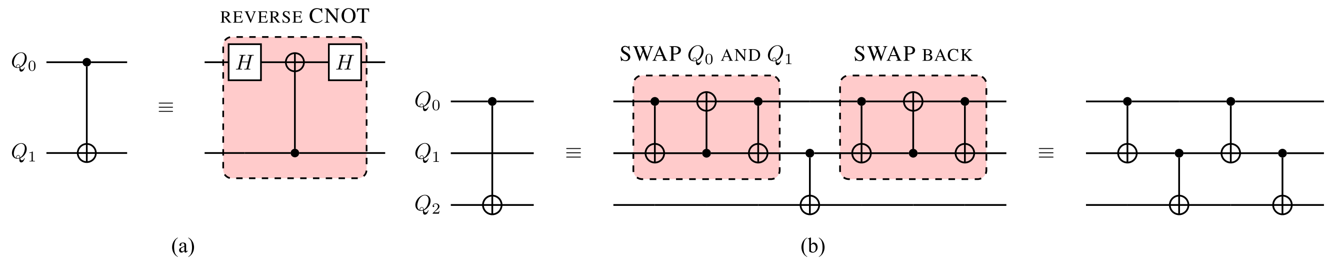


FIG. 3. **Example of equivalent quantum circuits generated during the quantum compilation for mapping an arbitrary CNOT into a sequence of CNOTs that can be directly executed by a given quantum processor. (a) Reversing CNOT [23]. A CNOT between Q_0 (control) and Q_1 (target) can be executed with the coupling map given in Fig. 2 by performing a CNOT between Q_1 (control) and Q_0 (target) sandwiched between two H gates. We note that the IBM Melbourne processor (shown in Fig. 2) natively supports CNOTs in both directions between neighbor qubits. (b) CNOT between qubits Q_0 (control) and Q_2 (target) can be executed through either: i) *quantum state transfer*, by first swapping qubits Q_0 and Q_1 so that Q_0 and Q_2 become adjacent qubits in the coupling maps, then by performing a CNOT between Q_0 and Q_2, and finally by swapping again qubits Q_0 and Q_1 so that they recover their initial position, or ii) *ancilla qubit*, by performing four CNOT operations between neighbor qubits with the help of the intermediate qubit Q_1.**

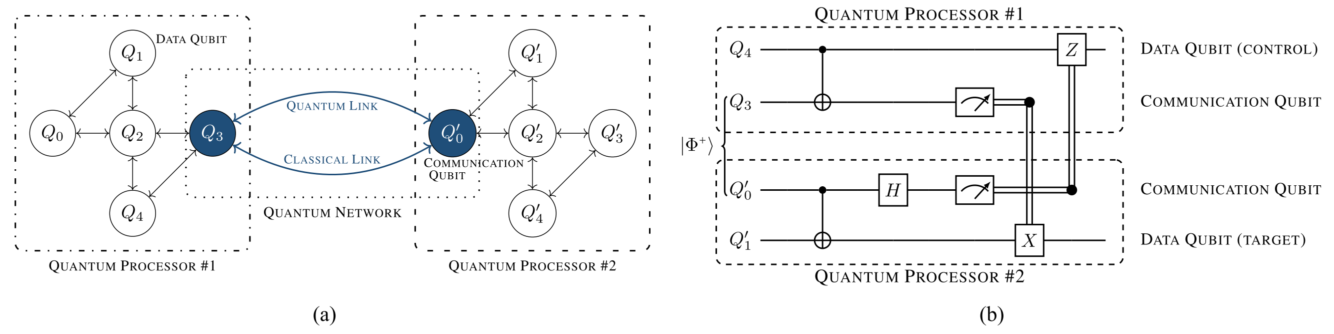


FIG. 4. **Toy-model for distributed quantum computing, with two quantum processors interconnected through a quantum network. (a) shows the network topology along with the processors coupling maps, whereas (b) provides the quantum circuit detailing the classical (2 bits) and the quantum (the Bell state) resources needed to execute a remote operation. (a) Two IBM Yorktown quantum processors are interconnected with a quantum link and a classical link. The classical link is used to transmit classical information, whereas the quantum link is needed to distribute *Bell states*—that is, maximally entangled two-qubit states—between remote processors to execute remote operations. Indeed, at least one physical qubit at each processor must be reserved for storing the Bell state, as discussed in (b). This kind of qubits—dark-blue-colored in the figure—are called *communication qubits* [3], [30] to distinguish them from the remaining physical qubits—white-colored in the figure—devoted to computing and referred to as *data qubits*. (b) Remote CNOT. To perform a CNOT between remote physical qubits stored at different processors—say data qubits Q_4 and Q'_1 in (a)—a Bell state such as Φ^+ must be distributed through the quantum link so that each pair member is stored within a *communication qubit* at each processor. Once the Bell state is available, the remote CNOT is obtained with a local CNOT between the *data* and the *communication qubit* at each processor, followed by a conditional gate on the data qubit depending on the measurement of the remote communication qubit. The double line denotes the transmission of 1 bit of classical information—i.e., the measurement output—between the remote processors.**

CNOT through a sequence of local CNOTs and single-qubit operations/measurements, as shown in Fig. 4(b).

To distribute Bell states between different quantum processors, at least one qubit at each processor—referred to as *communication qubit* [30] to distinguish it from the remaining *data qubits* devoted to processing—must be reserved for remote interprocessor operations. Hence, a crucial tradeoff between communication and data qubits arises. Specifically, for each remote CNOT, a Bell state is consumed [see Fig. 4(b)] and a new Bell state must be distributed between the remote processors through the quantum link before another remote CNOT can be executed. Hence, the more communication qubits are available within a processor, the more remote CNOTs can be executed in parallel, reducing the overhead induced by the distributed computation. But the more communication qubits are available for interprocessor communication, the less valuable resources—i.e., data qubits—are available for computing.

It is unlikely that a data qubit could be dynamically turned into a communication qubit during the compilation, given the dedicated hardware—such as a matter-flying qubit interface [21]—required for entanglement distribution. Conversely, it is reasonable to envision that the distributed quantum compiler could easily reserve—when multiple communication qubits are available at the same processor—a subset of the communication qubits for computing. This optimization task represents an interesting yet unaddressed open problem.

2) DYNAMIC CONNECTIVITY

As mentioned in Section II-A, with single-processor quantum computing, all the constraints on the possible interactions between different qubits—arising from the underlying physical computing architecture—can be effectively represented with a coupling map. Formally, a coupling map is a visual representation of the directed graph G

$$G = (\mathcal{V}, \mathcal{E}) \tag{1}$$

where $\mathcal{V} = \{v_i\}$ denotes the set of vertices representing the qubits and $\mathcal{E} = \{e_{i,j}\}_{v_i,v_j \in V}$ denotes the set of directed edges

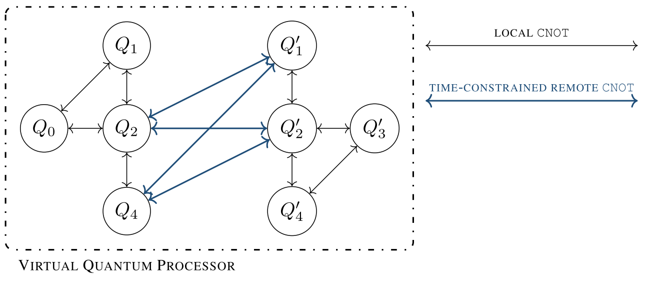


FIG. 5. Dynamic coupling map for the network topology shown in Fig. 4(a). The two 5-qubit quantum processors constitute an 8-qubit *virtual quantum processor* with qubits interconnected through both local and remote CNOTs. While the local CNOTs can be concurrently executed—i.e., they are *unconstrained*—the parallel execution of multiple remote CNOTs is *constrained* to the availability of multiple Bell states, with one Bell state for each concurrent remote CNOT. Given that only one communication qubit is available at each processor in Fig. 4(a), out of four remote CNOTs, only one can be executed at any time.

representing the possibility to perform[4] a CNOT, with v_i and v_j acting as control and target qubit, respectively.

But when it comes to distributed quantum computing, a new kind of constraints arises as a consequence of the underlying physical network topology.

More in detail, similarly to single-processor quantum compiling, the remote operations are restricted to certain fixed pairs. Specifically, they are restricted to pairs composed of data qubits directly connected to a communication qubit within the processor coupling map. For instance, with reference to the network topology shown in Fig. 4(a), a remote CNOT between data qubits Q_2 and Q'_1 can be directly mapped onto the circuit given in Fig. 4(b). Conversely, a remote CNOT between data qubits Q_2 and Q'_4 cannot be directly executed between the pair, but it requires to distribute the operation through the neighbor qubits, as shown in Fig. 3(b).

However, differently from single-processor compiling, the remote operations are subjects to two types of *temporal constraints*.

1) *Simultaneity Limitations:* As previously discussed, each remote CNOT relies on the availability of a Bell state stored within a communication qubit. Hence, even if remote CNOTs can—in principle—be executed between different remote pairs, the number of remote CNOTs that can be executed simultaneously between two processors is limited by the number of communication qubits jointly available at each processor. With reference to Fig. 5, out of four possible remote CNOTs (denoted with blue arrows), only one can be executed at any time.
2) *Consecutiveness Limitations:* Each remote CNOT consumes a Bell state as a consequence of the measurement operations on the communication qubits [21]. Accordingly, a new Bell state must be generated and distributed through the quantum link to the communication qubits, before a subsequent remote CNOT could be executed. And even if the Bell state distribution can start right after the measurements, it is reasonable to assume—given the several order of magnitudes separating intraprocessor qubit distance from interprocessor one—that the time needed to entangle the communication qubits significantly exceeds the time required for local CNOTs.[5] Accordingly, we have two major issues. First, the "clock" of the remote operations will be significantly lower than the "clock" of the local operations, and hence, it becomes fundamental to minimize the number of remote—rather than the number of local—operations to preserve the quantum information integrity from decoherence (see Section II-A). Furthermore, there may be periods of time—following the execution of a remote operation up to the successful distribution of a new Bell state—during which the quantum processors are *disconnected* and only local operations are possible.

These additional constraints must be properly modeled within the coupling map, so that the distributed quantum compiler can optimize the quantum circuit by accounting for the temporal dynamics arising with the distributed architecture. And this represents an open problem.

3) AUGMENTED CONNECTIVITY

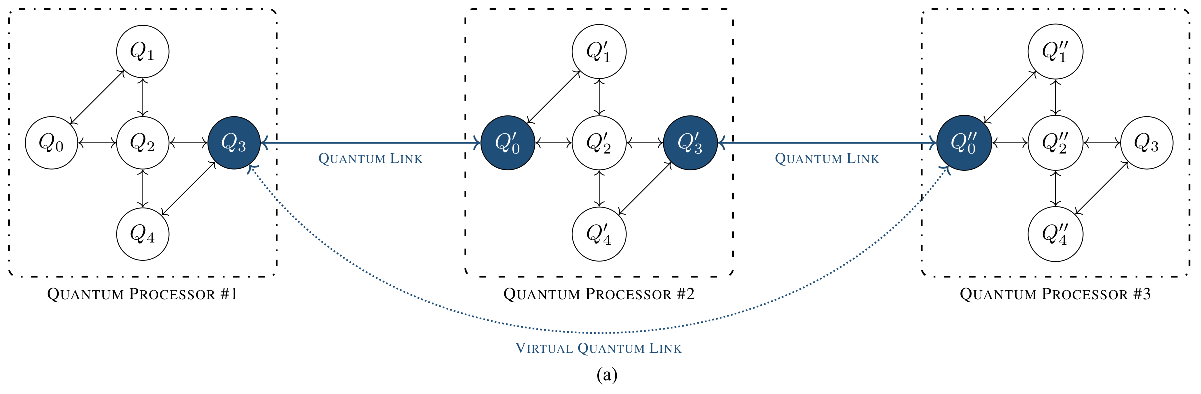
As shown in Fig. 3(b), single-processor quantum computing must resort to either *state transfer* (swapping) or *ancilla* strategy to implement a CNOT between nonadjacent (within the coupling map) physical qubits. The rationale for this lays in the impossibility to have direct interactions between distant qubits. And the further the qubits are within the coupling map, the longer the sequence of additional CNOTs is required, regardless of the adopted strategy.

Conversely, distributed quantum computing can exploit a strategy—called *entanglement swapping* [4] and summarized in Fig. 6—to implement a remote CNOT between qubits stored at remote processors, even if the processors are not directly connected through a quantum link.

In a nutshell, to distribute a Bell state between remote processors—say quantum processor #1 and #3 in Fig. 6(a)—two Bell states must be first distributed through the quantum links so that one Bell state is shared between the first processor and an intermediate node and another Bell state is shared by the same intermediate node and the second processor. Then, by performing a Bell state measurement (consisting of an H and a CNOT gate, followed by a joint measurement) on the communication qubits at the intermediate node—i.e., qubits Q'_0 and Q'_3 in Fig. 6(b)—a Bell state is obtained at the remote communication qubits—i.e., qubits Q''_0 and Q_3 in

[4]In the following, for the sake of presentation, we consider the simplest binary case, i.e., either the operation can or cannot be executed. But the discussions, as well as the results derived in the following, continue to hold when a weight—usually representing the gate fidelity—is mapped on the edge.

[5]In the order of hundreds of nanoseconds for the IBM Yorktown quantum processor [31].



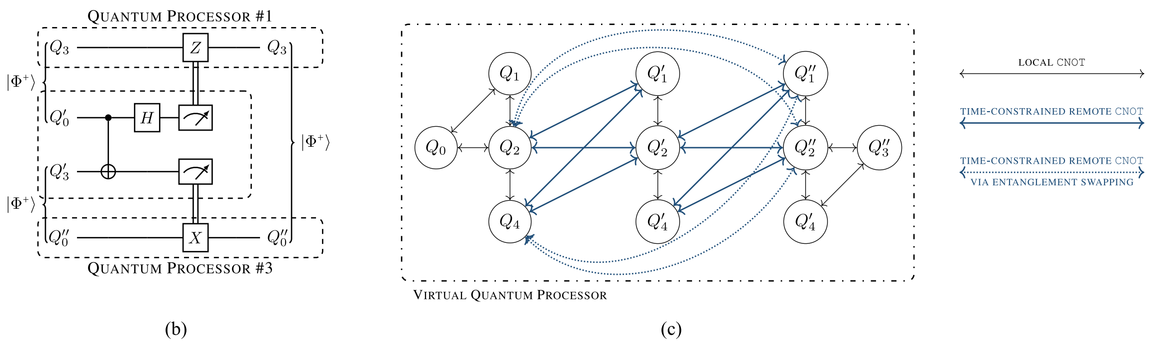

QUANTUM PROCESSOR #1
|Φ⁺⟩
|Φ⁺⟩
|Φ⁺⟩
Q₃
Q′₀
Q′₃
Q″₀
H
Z
X
QUANTUM PROCESSOR #3
(b)
VIRTUAL QUANTUM PROCESSOR
(c)
LOCAL CNOT
TIME-CONSTRAINED REMOTE CNOT
TIME-CONSTRAINED REMOTE CNOT VIA ENTANGLEMENT SWAPPING

FIG. 6. Augmented connectivity. Entanglement swapping increases the connectivity between physical qubits, with a number of possible remote CNOTs that scales at least linearly with the number of processors. (a) By swapping the entanglement at the intermediate nodes—namely, quantum processor #2—it is possible to distribute a Bell state between remote processors—namely, processors #1 and #3—even if they are not adjacent, i.e., they are not directly connected through a quantum link. Hence, entanglement swapping enhances the network connectivity through *virtual quantum links.*
(b) Entanglement swapping. A Bell state can be distributed between remote processors by swapping the entanglement at an intermediate node through local processing and classical communication. (c) Dynamic coupling map for the network topology shown in Fig. 6(a). The solid blue lines denote remote CNOTs between adjacent processors, whereas the dotted blue lines denote remote CNOTs between distant processors achievable via entanglement swapping.

Fig. 6(a)—by applying some local processing at the remote nodes depending on the (classical) output of the Bell state measurement.

From the above, it becomes clear that entanglement swapping significantly increases the connectivity within the virtual quantum processor. As an instance, qubit Q_4 in Fig. 6(a) can interact with just two qubits within the same processor via local CNOTs and two qubits within the neighbor processor via remote CNOTs. However, it can interact with two more qubits—i.e., Q''_1 and Q''_2—via entanglement swapping. And the higher is the number of available quantum processors, the higher is the number of possible interactions. Indeed, the number of additional interactions via entanglement swapping scales linearly with the number of available processors when only two communication qubits are available at each intermediate processor. If this constraint is relaxed, the number of additional interactions via entanglement swapping scales more than linearly.

However, it must be acknowledged that the augmented connectivity provided by entanglement swapping does not come for free. Indeed, entanglement swapping consumes the Bell states stored within the communication qubits at the intermediate processors. And the higher the number of intermediate processors, the higher the number of consumed Bell states.

Hence, a tradeoff between "augmented connectivity" and "EPR cost" arises with entanglement swapping, and a distributed quantum compiler must carefully account for these pros and cons.

B. RELATED WORK

Most quantum computer proposals are based on variations of the nearest-neighbor, two-qubit, and concurrent execution (NTC) architecture [32]. Depending on the layout of qubits, there are three NTC architectures: 1-D, 2-D, and 3-D. The 1-D model, called linear nearest neighbor (LNN) [33], consists of qubits located in a single line. In this model, only two neighboring qubits can interact. This is the most challenging scenario. The effects of the LNN model on performance have been investigated for many relevant use cases, such as the quantum Fourier transform [34], [35], Shor's algorithm [36], [37], and adders [38].

Beals *et al.* [39] provided algorithms for efficiently moving and addressing quantum memory in parallel. These imply that the standard circuit model can be simulated with low overhead by a more realistic model of a distributed quantum computer. The authors show that for an LNN N-qubit architecture, $O(N)$ time steps are necessary for performing $N/2$ two-qubit gates in parallel. However, it is worthwhile to note that the developed analysis does not consider any

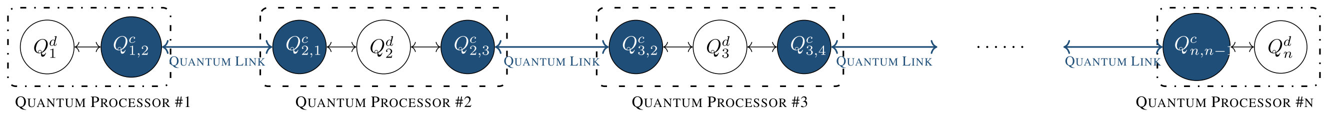


FIG. 7. Worst-case scenario in terms of overhead induced by the distributed computation: the quantum processors are interconnected through a 1-D nearest-neighbor topology, and only one data qubit is available at each quantum processor. Intraprocessor coupling between communication qubits is omitted for the sake of simplicity.

additional overhead induced by the compilation task, and the derived Big-O bound relies on linear constant that is in the “many, many thousands” [40]. Conversely, in the following, we develop an analysis that explicitly considers the additional overhead induced by the compilation task, as discussed in Section I.

Zomorodi-Moghadam *et al.* [10] proposed a general approach, based on the Kernighan–Lin algorithm for graph partitioning, to optimize the number of teleportations for a distributed quantum computing architecture consisting of two spatially separated and long-distance quantum subsystems. The same authors also proposed an approach based on dynamic programming [41].

Andrés-Martínez and Heunen [42] proposed an approach that may distribute circuits across any number of quantum devices. The main idea is to turn the quantum circuit into a hypergraph and then find a partitioning that minimizes the number of cuts, as each cut corresponds to a Bell state shared across two QPUs by means of communication qubits. The partitioning problem is addressed by means of the KaHyPar solver [43]. The proposed solution has some drawbacks, in particular that there is no way to define the number of communication qubits of each QPU. In the software implementation of the algorithm, the number of available communication qubits is unlimited and cannot be constrained.

IV. COMPILER DESIGN AND OVERHEAD BOUNDS

As discussed in Section III, several additional constraints arise with the shift from single processor to distributed quantum compiling. Given that single-processor quantum compiling has already been proved to be NP-complete [14], it is reasonable to expect that optimal distributed quantum compiling is an even harder challenging task.

For this reason, in the following, we take a completely different approach. Specifically, we aim at designing a *general-purpose*, *efficient*, and *effective* compiler for distributed quantum computing.

General-purpose because our compiler does not require any particular assumption on the quantum circuit to be compiled.

Efficient because—as proved in Section V-A—our compiler is computationally efficient, exhibiting a polynomial time complexity that grows polynomially with the number of logical qubits and linearly with the depth of the quantum circuit to be compiled.

Effective because—as proved in Section IV-B—our compiler assures a polynomial worst-case overhead, in terms of both depth of the compiled quantum circuit and number of calls to the costliest and most challenging task, i.e., the link entanglement generation.

A. SYSTEM MODEL

We consider the worst-case scenario shown in Fig. 7. More in detail, we assume that only one data qubit is available at each quantum processor.[6] The rationale for this choice is as follows. Whenever multiple data qubits are available at a single quantum processor, a local `CNOT` can be executed between these data qubits without incurring in any overhead induced by the distributed computation. Conversely, with just one data qubit available at each processor, each and every `CNOT` within the quantum circuit must be mapped into a remote `CNOT`, and hence, the overhead induced by the distributed computation is the highest possible.

Furthermore, we assume that the quantum processors are interconnected through a 1-D nearest-neighbor topology, as shown in Fig. 7. Again, the rationale for this choice is to consider the worst-case scenario in terms of overhead induced by the distributed computation. In fact, the considered topology is characterized by the lowest possible number of communications qubits—i.e., $2n - 2$, with n denoting the number of quantum processors—since the removal of any communication qubit would disconnect the network into two disjoint subsets of quantum processors. And the quantum processors are arranged in a line—rather than in a star—to maximize both the number of nonadjacent quantum processors and the maximum distance—in terms of *hops*—between two nonadjacent quantum processors.

From the above, it becomes clear that the considered architecture represents the worst-case scenario in terms of overhead induced by the distributed computation. Hence, the *actual* overhead induced by any real-world architecture will be always upper-bounded by the communication overhead induced by the considered architecture.

Clearly, we need to choose a metric for measuring the overhead induced by the distributed computation. As discussed in Section II-A, there exists a general consensus on circuit depth as a key performance metric of circuit compilation. Hence, in the following, we measure the overhead in terms of *number of additional layers required to distribute the computation of a single layer in the original quantum circuit*. Furthermore, we also evaluate the overhead in terms of how many calls to the link entanglement generation process are required from the compiling algorithm.

[6]Clearly, the total number of data qubits within the distributed architecture must be greater than the number of logical qubits within the quantum circuit to be compiled.

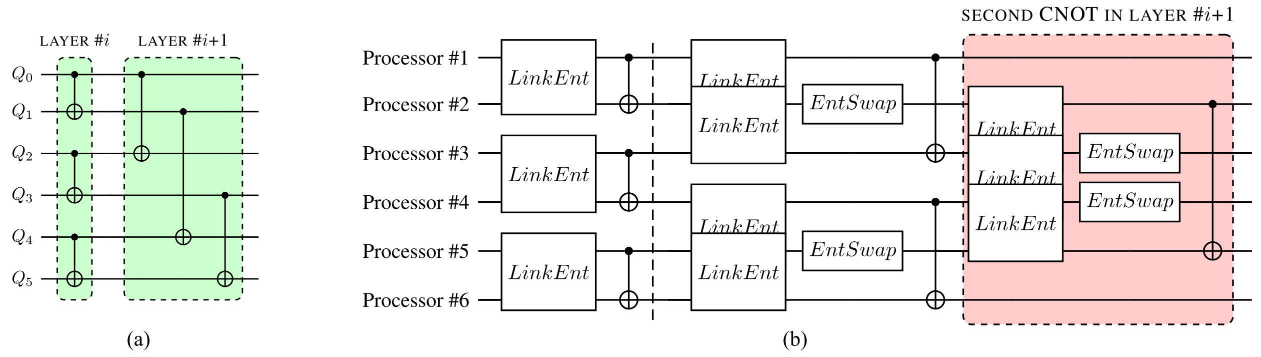


FIG. 8. ***Entanglement swapping strategy*. Each remote CNOT in (a) requires two preliminary tasks: 1) *link entanglement*, for distributing the entanglement between neighbor nodes and 2) *entanglement swapping*, for entangling the two remote processors involved within the CNOT. Clearly, the swapping task is omitted whenever the CNOT operates between data qubits stored at processors that are neighbors within the network topology, as for the *i*th layer. (a) Original quantum circuit. (b) Compiled quantum circuit.**

B. BASIC STRATEGIES FOR DISTRIBUTING CNOTS

Let us consider a single layer of the original n-qubit quantum circuit. Clearly, the number of CNOTs in each layer is lower or equal to $\frac{n}{2}$, given that at most $\frac{n}{2}$ gates can be executed simultaneously (and thus belong to the same layer) by operating on different pairs of qubits.

As discussed in Section IV-A, we aim at considering the worst-case scenario in terms of overhead induced by the distributed computing architecture. Hence, each CNOT within the quantum circuit—given that it operates on physical qubits stored at different processors, as discussed in Section IV-A—is a remote CNOT. As a consequence, the compiler must map at most $\frac{n}{2}$ remote CNOTs in each layer.

1) ENTANGLEMENT-SWAPPING-BASED STRATEGY

The first strategy for implementing remote CNOTs is based on the *entanglement swapping* technique discussed in Section III-A and shown in Fig. 6.

Accordingly, each remote CNOT is implemented by first generating link entanglement [44] among neighbor nodes. To this aim, different techniques for entanglement generation can be employed, depending on the particulars of the underlying qubit technology [21]. Nevertheless, link entanglements can be simultaneously generated, given that each processor is equipped with two communication qubits. Once generated, the entanglement is simultaneously[7] swapped at intermediate nodes so that a Bell state is distributed between the two remote processors, and, finally, the remote CNOT is obtained, as shown in Fig. 4(b).

The *entanglement-swapping-based strategy* is outlined in Fig. 8(b) in terms of basic tasks. Within the figure, the particulars of each task are omitted for the sake of clarity. For instance, entanglement swapping—although depicted as a single block—is indeed obtained with a quantum circuit composed by three layers, as shown in Fig. 6(b). Similarly, the link entanglement generation requires a quantum circuit with a depth equal or greater than two, depending on the particulars of the quantum technology underlying entanglement generation and distribution [21].

Nevertheless, the figure[8] provides a clear intuition of both the sequentiality constraints between the different tasks and the parallelism achievable within each task. Specifically, whenever the CNOTs *overlaps*[9] within the network topology [as for the CNOTs of the layer #i+1 in Fig. 8(a)], they must be executed sequentially. Differently, CNOTs that do not overlap [as for the CNOTs of the ith layer in Fig. 8(a)] can be executed simultaneously. Since we are interested in assessing the worst-case overhead induced by distributed computation, in the following, we consider the worst-case scenario, in which all the CNOTs of an arbitrary layer of the quantum circuit *overlap* within the network topology. Hence, we have that the depth overhead of the *entanglement-swapping-based strategy* does not exceed the following depth:

$$\frac{n}{2} d_{es} \tag{2}$$

where n denotes the number of logical qubits within the quantum circuit and d_{es} is a constant factor (independent from the characteristics of the original quantum circuit) given by

$$d_{es} = c_{le} + c_{bsm} + c_{cx} \tag{3}$$

[7]In general, the capability to generate (and to regenerate, once depleted) and distribute entangled Bell states through different links in parallel depends on the quantum resources available, i.e., both the number of communication qubits at each processor and the interconnection (shared bus versus point-to-point) among the communication qubits. Differently, the possibility to simultaneously swap the entanglement at the intermediate nodes depends only on classical resources, i.e., the possibility to simultaneously transfer classical information.

[8]We note that—for the sake of simplicity—in Fig. 8(b), we simply mapped the jth logical qubit Q_j of layer #i in Fig. 8(a) onto the $(j+1)$th processor, ignoring so any optimization achievable with a proper mapping of the logical qubits of the quantum circuit onto the physical qubits of the quantum processor.

[9]The term "*overlap*" indicates the case when the execution of the considered CNOTs involves overlapping sets of intermediate processors as a consequence of the constraint we imposed on the network topology of having $2n-2$ communication qubits. With reference to the example in Fig. 8(a), the CNOT between Q_0 and Q_2 in the layer #i+1 overlaps with the CNOT between Q_1 and Q_4, being the communication qubits at the processors #1 and #2 needed to both of them. Differently, the CNOT between Q_3 and Q_5 does not overlap with the CNOT between Q_0 and Q_2, and hence, they can be performed in parallel.

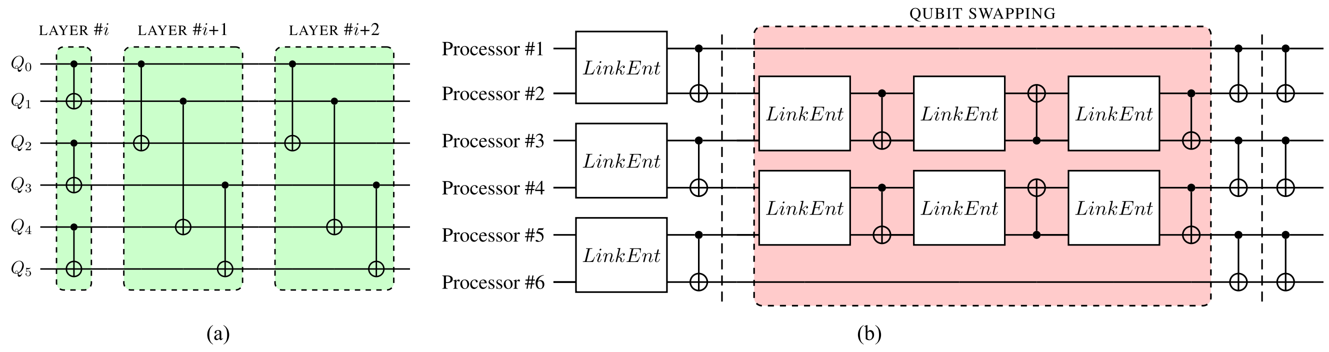


FIG. 9. ***Data-qubit-swapping-based strategy*. Swapping data qubits between remote quantum processors can be advantageous whenever the original quantum circuit presents repetitions of the same CNOT interaction pattern, as for layers #*i*+1 and #*i*+2 in (a). Although not shown in the figure, the entanglement swapping tasks [as in Fig. 8(b)] are needed whenever it is necessary to swap data qubits stored at processors that are not neighbors within the network topology. (a) Original quantum circuit. (b) Compiled quantum circuit.**

with c_{le} and c_{bsm} denoting the number of layers required to perform the link entanglement task and the entanglement swapping task, respectively, and c_{cx} denoting the number of layers required to perform a remote CNOT once the Bell state has been distributed between two processors. The actual values of c_{le}, c_{bsm}, and c_{cx} depend on the particulars of the underlying hardware technology.

From (2), we have that the actual depth of an arbitrary d-depth quantum circuit compiled with the *entanglement swapping based strategy* will always be lower than $\frac{n}{2}\,d$, neglecting the constant d_{es}. Hence, the depth overhead grows linearly with the number of logical qubits of the quantum circuit to be compiled. Given that this result holds for the worst-case scenario (one-data-qubit processors arranged in a one-dimensional network topology), the actual depth overhead induced by any arbitrary distributed architecture will always be upper-bounded by (2).

We further note that classical information must be exchanged between the quantum processors. For instance, the entanglement swapping task requires the transmission of classical information (i.e., the measurement output) throughout the quantum network. Hence, in case of long-distance quantum processors, the actual execution time of the compiled quantum circuit may be affected by the latency induced by the classical communications.

Finally, due to the complex and stochastic nature of the physical mechanisms underlying quantum entanglement [44], several attempts can be required for establishing a link entanglement, and this may also impact the execution time of the compiled quantum circuit. Indeed, we should consider link entanglement as the *critical task* for distributed quantum computation, given that the remaining tasks require only local quantum operations and classical communications. From this perspective, the *entanglement-swapping-based strategy* requires at most $\frac{n}{2}$ repetitions of the link entanglement task, regardless of the original quantum circuit and regardless of the characteristics of the network topology underlying the distributed computing architecture.

2) DATA-QUBIT-SWAPPING-BASED STRATEGY

The *entanglement-swapping-based strategy* takes full advantage of the augmented connectivity enabled by the communication qubits—as discussed in Section III-A—to allow interactions between remote processors within each layer.

Nevertheless, whenever the original quantum circuit presents repetitions of the same CNOT interaction pattern between logical qubits—as for layers #*i*+1 and #*i*+2 in Fig. 9(a)—a more elaborate strategy—based on *moving* the data qubits—can provide better performance.

The strategy is shown in Fig. 9: the objective is *to arrange* (i.e., to swap) the data qubits within the quantum processors so that eventually each CNOT of the original layer operates on qubits stored at neighbor processors within the network topology.

Intuitively, the strategy goal can be modeled as an array sorting problem. Indeed, similarly to classical sorting, the n data qubits (representing the *values* to be sorted) must be ordered within the network topology (representing an array with size equal or greater than n). However, differently from classical sorting where any couple of values can be swapped regardless from their position within the array, with the *data-qubit swapping*, the constraints arising from the underlying network topology must be carefully taken into account. To this aim, by taking advantage of the *sorting network theory*, it is easy to model the network topology constraints through the notion of *insertion network* (or, equivalently, *bubble network*). As a consequence, the overall depth of the equivalent quantum circuit grows with the number n of logical qubits as [45]

$$2n-3 \tag{4}$$

instead of a logarithmic $\log n$ depth factor as for classical sorting.

Nevertheless, sorting networks—and in general classical sorting—are based on the assumption that there exists a total (monotonic) order over the array elements. Hence, there exists a unique solution to the sorting problem. Conversely, the *data-qubit-swapping-based strategy* admits several equivalent solutions for the arranging problem, as exemplified in

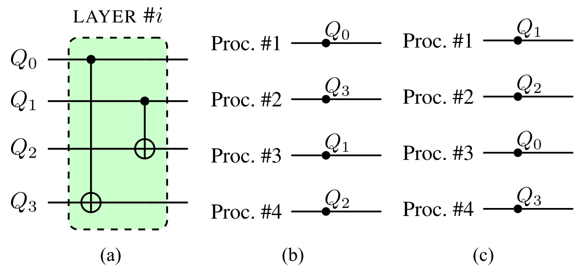


FIG. 10. ***Data-qubit-swapping-based strategy:*** **equivalent mappings. Both (b) and (c) represent valid arrangements of the data qubits within the remote quantum processors so that each** CNOT **in (a) operates on qubits stored at processors neighbor within the network topology. (a) Original quantum circuit. (b) Possible arrangement. (c) Alternative equivalent arrangement.**

Algorithm 1: Data-Qubit Swapping.
Input: n-qubit circuit layer L with mod(n,4) $=0$ and $\frac{n}{2}$CNOT*s*
Output: layer L with each CNOT Operating on Neighbor Qubits.

```
1:  function Sort ($L$)
2:      if $\exists$ CNOT($q_i, q_j$) with $i, j \leq \frac{n}{2}$ then
3:          //$\exists$ CNOT($q_k, q_l$) with $k, l > \frac{n}{2}$
4:          SWAP($q_{i+1}, q_j$)
5:          SWAP($q_{k+1}, q_l$)
6:          $L = L \setminus \{q_i, q_{i+1}, q_k, q_{k+1}\}$
7:      else
8:          // $\exists$ CNOT($q_{\frac{n}{2}}, q_l$) with $l > \frac{n}{2}$
9:          // and $\exists$ CNOT($q_i, q_{l-1}$) with $i < n/2$
10:         SWAP($q_{\frac{n}{2}}, q_{l-1}$)
11:         SWAP($q_i, q_{\frac{n}{2}-1}$)
12:         $L = L \setminus \{q_{\frac{n}{2}-1}, q_{\frac{n}{2}}, q_{l-1}, q_l\}$
13:     end if
14:     if $L \neq \emptyset$ then
15:         Sort($L$)
16:     end if
17: end Function
```

Fig. 10. We now formalize these considerations with the following theorem.

Theorem 1: Let us consider the ith layer of an arbitrary n-qubit quantum circuit. The depth of the corresponding compiled quantum circuit, obtained through the *data-qubit-swapping-based strategy*, does not exceed the following depth:

$$\frac{n}{4} d_{qs} + d'_{qs} \tag{5}$$

where d_{qs} and d'_{qs} are constant factors (independent from the characteristics of the original quantum circuit) given by

$$d_{qs} = 3\,(c_{le} + c_{bsm} + c_{cx}) \tag{6}$$

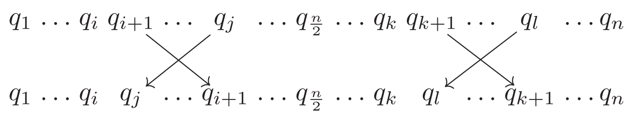
$$d'_{qs} = c_{le} + c_{cx} \tag{7}$$

with c_{le} and c_{bsm} denoting the number of layers required to perform the link entanglement task and the entanglement swapping task, respectively, and c_{cx} denoting the number of layers required to perform the remote CNOTs once the Bell state has been distributed between two processors.

Proof: The proof easily follows by recognizing that: 1) the function SORT$(\cdot)$, defined in Algorithm 1, is called at most $\frac{n}{4}$ times; and 2) after these calls to SORT$(\cdot)$, all the CNOTs, by acting on qubits stored at neighbor processors, can be executed at once through link entanglement followed by local operations, as shown in Fig. 4(b).

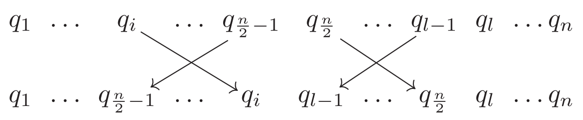
More specifically, in each call, we have two disjoint cases (line 2).

In the former case (lines 3–6), there exists a CNOT acting within the first *half portion*—i.e., the first $\frac{n}{2}$ logical qubits—of the original layer. Since we are considering the worst case—namely, a layer with $\frac{n}{2}$ CNOTs—then we have that there exists at least one CNOT acting on the last *half portion*—i.e., the last $\frac{n}{2}$ logical qubits—of the original layer. Hence, the two CNOTs do not overlap and two simultaneous SWAP operations can be executed—one in each *half portion* of the original quantum circuit—as shown here:


$q_1 \cdots q_i\ q_{i+1} \cdots\ q_j\ \cdots q_{\frac{n}{2}} \cdots q_k\ q_{k+1} \cdots\ q_l\ \cdots q_n$
$q_1 \cdots q_i\ q_j \cdots q_{i+1} \cdots q_{\frac{n}{2}} \cdots q_k\ q_l \cdots q_{k+1} \cdots q_n$

so that, within the compiled circuit—the two CNOTs act on qubits stored at neighbor processors. Within the previous diagram, as well as in Algorithm 1, we omitted some minor particulars for the sake of simplicity. For instance, we implicitly assumed that i (and k) is odd—i.e., mod$(i, 2) = 1$—so that q_j must be swapped with q_{i+1}. Clearly, whether i should be even, q_j must be swapped with q_{i-1}.

In the latter case (lines 8–12), each and every CNOT acts on two logical qubits belonging to both the *half portions* of the original layer. Let us consider, with no lack of generality, the CNOT acting on the $\frac{n}{2}$th logical qubit (i.e., the last qubit of the first *half portion*) and let us denote as q_l the second qubit on which such a CNOT operates. Since we are considering a layer with $\frac{n}{2}$ CNOTs, then we have that there exists a CNOT acting on the $l-1$th qubit. By denoting as q_i the second qubit on which such a CNOT operates, it follows $i < \frac{n}{2}$. Although the two CNOTs overlap, by properly selecting two SWAP operations, as shown here:


$q_1\ \cdots\ q_i\ \cdots\ q_{\frac{n}{2}-1}\ q_{\frac{n}{2}}\ \cdots\ q_{l-1}\ q_l\ \cdots q_n$
$q_1\ \cdots\ q_{\frac{n}{2}-1}\ \cdots\ q_i\ q_{l-1}\ \cdots\ q_{\frac{n}{2}}\ q_l\ \cdots q_n$

we have that the SWAPs can be simultaneously executed so that—within the compiled circuit—the two CNOTs act on qubits stored at neighbor processors. Regardless of which case holds, each call to the *SORT*$(\cdot)$ function compiles two CNOTs. By recalling that at most $\frac{n}{2}$ CNOTs are present in a layer, the thesis follows. ■

From (5), we have that the actual depth of an arbitrary d-depth quantum circuit compiled with the *data-qubit swapping based strategy* will always be lower than $\frac{n}{4}\,d$, neglecting the constant factors. Hence, the depth overhead is asymptotically lower than the overhead induced by the *entanglement-swapping-based strategy*. However, an explicit comparison between the two strategies depends on the particulars of the underlying qubit technology through the exact expressions of d_{es} and d_{qs}. Furthermore, it also depends on the repetitions of the same CNOT interaction patterns within the original quantum circuit.

As regards the number of repetitions of the link entanglement task, in general, it depends on the characteristics of the underlying qubit technology, as discussed in Section IV-C. With reference to the IBM quantum processors, where a SWAP operation is obtained through a sequence of three CNOTs, from (5)–(7), we have that the *data-qubit-swapping-based strategy* requires at most $2n$ repetitions of the link entanglement task, regardless of the original quantum circuit and regardless of the characteristics of the network topology underlying the distributed computing architecture.

C. DISCUSSION

As already mentioned above, the performance of the two strategies firmly depends on the particulars of the underlying hardware technology through the parameters d_{es}, d_{qs}, and d'_{qs} given in (3), (6), and (7).

To better clarify this point, let us consider d_{qs}, which inherently denotes the cost for a SWAP operation. Having assumed in this article the CNOT being the fundamental multiqubit gate, a single remote SWAP operation can be obtained through three remote CNOTs, as in Fig. 3(b). And this is the rationale for the constant factor equal to 3 in (6), which accounts for the cost of three remote CNOTs.

Clearly, by changing the assumptions on the underlying hardware technology, the expression of d_{qs} changes as well. For instance, photonic technology can provide the SWAP gate as the native operation [46], and in such a case, d_{qs} is equal to 1. Nevertheless, the main result—i.e., (5)—continues to hold. Furthermore, whenever the SWAP gate is the native operation, a single CNOT can be obtained through two consecutive SWAPs interleaved by single-qubit operations [47]. Hence, the expression of d_{es} must change accordingly but the main result—i.e., (3)—continues to hold as well.

Indeed, it is worthwhile to note that, despite the differences between the performance of the two strategies, there exists a one-to-one mapping between the strategies. Specifically, there exists an admissible transformation allowing to map the compiled circuit obtained with a strategy into the compiled circuit obtained with the other strategy. And the corresponding computational task exhibits a polynomial-time complexity[10] for every original circuit.

[10]Given that both the strategies exhibit a polynomial-time computational complexity, as proved in Section V-A.

V. PERFORMANCE ANALYSIS

Here, we perform a performance analysis for the compiler design conducted in Section IV. More in detail, in Section V-A, we illustrate the algorithmic implementation of the compiler, proving so its attractive feature—*a polynomial time complexity that grows quadratically with the number of logical qubits and linearly with the depth of the quantum circuit to be compiled*—from a computational perspective. Then, in Section V-B, we validate the theoretical upper bounds on the number of layers that result from compiling a layer of remote CNOTs, derived in Section IV-B, against an extensive set of medium-size quantum circuits of practical interest. Finally, with Sections V-C and V-D, we conclude the performance analysis through an *unfair*—as clearly shown with Fig. 14—comparison with the state of the art for two different network topologies.

A. COMPILER IMPLEMENTATION

We implemented the strategies discussed in Section IV-B in Python, using Qiskit [27] as the development framework. Given a quantum circuit described in the QASM format, the compiler proceeds to instantiate a distributed architecture that mimics the one described in Section IV-A. To model the worst-case scenario depicted in Fig. 7, each QPU has one data qubit and two communication qubits. Moreover, each QPU has two neighbor QPUs, with the exception of the outer QPUs that have one neighbor QPU only. Each qubit of the circuit is assigned to the data qubit of one QPU.

The compilation process is summarized and illustrated in Fig. 11. Reading the circuit from left to right, the *front layer*, *i.e.*, a layer comprising only CNOT gates that can be executed in parallel, is updated. To this end, one-qubit gates are immediately mapped to the compiled circuit, while CNOT gates are added to the front layer. This is done until all logical qubits are interested by a CNOT or there are no more CNOT gates that can be executed in parallel in the current front layer. Then, the front layer is compiled, rendering all currently involved CNOT gates executable. This process is repeated until no more front layers can be computed, meaning that all the circuit gates have been mapped to the distributed architecture.

Front layer compilation is based on Algorithm 2, where one can choose between the two strategies discussed in Section IV-B. When the *entanglement-swapping-based strategy* is adopted, remote CNOT gates preceded by entanglement swapping are applied whenever the involved QPUs are not neighbors. Note that to perform entanglement swapping, as well as remote CNOTs, we need to generate link entanglement between all involved QPUs.

Regarding the more advanced *data-qubit-swapping-based strategy*, the list representing the interaction between qubits is prepared, and then, Algorithm3 is used to compute the data swap operations needed to reorder the qubits. Referring to Fig.12(a), the list representing the interaction between qubits before swapping would be [112323], while the sorted list

Algorithm 2: COMPILEFRONTLAYER.
Input: the Front Layer $\mathcal{F}$ to be Compiled, the Executed Gates $\mathcal{E}$ until now
Output: the Executed Gates $\mathcal{E}$ Updated With the Compiled Front Layer $\mathcal{F}$.

```
if data-qubit-swapping-based strategy then
  prepare interactions vector
  swaps ← SORTPAIRS(INTERACTIONS)
  for all swap ∈ swaps do // Perform remote SWAPs
    if two EPR pairs between QPUs then
      Teleport(swap.q1, swap.q2)
    else
      REMOTECNOT(swap.q1, swap.q2)
      REMOTECNOT(swap.q2, swap.q1)
      REMOTECNOT(swap.q1, swap.q2)
    end if
  end for
end if
for all gate ∈ $\mathcal{F}$ do
  REMOTECNOT(gate.control, gate.target)
end for
return $\mathcal{E}$
function REMOTECNOT(control, target)
  if control and target are not on neighboring QPUs then
    $qpu_0$ ← QPUs of control
    $qpu_n$ ← QPUs of target
    for $i \in \{0, .., n-2\}$ do
      add link entanglement between $qpu_i$ and $qpu_{i+1}$ to $\mathcal{E}$
    end for
    add entanglement swap between $qpu_0$ and $qpu_n$ to $\mathcal{E}$
  end if
  add link entanglement between $qpu_0$ and $qpu_n$ to $\mathcal{E}$
  add a remote CNOT between control and target to $\mathcal{E}$
end Function
function TELEPORT(q1, q2)
  if q1 and q2 are not on neighboring QPUs then
    $qpu_0$ ← QPUs of q1
    $qpu_n$ ← QPUs of q2
    for $i \in \{0, .., n-2\}$ do
      add link entanglement between $qpu_i$ and $qpu_{i+1}$ to $\mathcal{E}$
        using two available EPR pairs between QPUs
    end for
    $e_{01}, e_{02}$ ← the two communication qubits at $qpu_0$
    $e_{n1}, e_{n2}$ ← the two communication qubits at $qpu_n$
    add entanglement swap between $e_{01}$ at $qpu_0$ and $e_{n1}$ at $qpu_n$ to $\mathcal{E}$
    add entanglement swap between $e_{02}$ at $qpu_0$ and $e_{n2}$ at $qpu_n$ to $\mathcal{E}$
    add quantum teleportation between q1 and $e_{n1}$ to $\mathcal{E}$
    add quantum teleportation between q2 and $e_{02}$ to $\mathcal{E}$
    add local swap between q1 and $e_{02}$
    add local swap between q2 and $e_{n1}$
  end if
end Function
```

after swapping would be [112233], meaning that no overlapping CNOTs are left in the layer.

The *data-qubit swapping* routine operates on lists with a number of elements—i.e., a number of logical qubits—that is a multiple of 4. For this reason, a couple of dummy values, set to -1, may have to be added to the end of the list whenever the number of CNOTs is odd. Given that the algorithm searches for swaps form left to right, the dummy couple at the end will be left untouched. After creating *masks* to keep track of already swapped qubits, the algorithm finds all necessary swaps in exactly $\frac{n}{4}$ steps, where n is the number of elements in the list, i.e., the number of logical qubits interested by CNOTs.

Taking into account the topology of the worst-case scenario shown in Fig. 7, to perform a swap, at least three remote CNOTs are needed. After all the data-qubit swaps have been applied, the necessary remote CNOTs have to be placed. Ideally, this last step would involve only remote CNOTs between neighbor QPUs, but when not all qubits of the circuit are involved in the front layer, such as Q_1 in Fig. 12(a), it may be still necessary to perform some entanglement swapping operations, as shown in Fig. 12(b). This is because, when sorting pairs, our algorithm does not take into account QPUs that are not involved in the current front layer. Consequently, the implementation of the *data-qubit-swapping-based strategy* is actually a hybrid between the two strategies described in Section IV-B, avoiding data-qubit swapping if not necessary and resorting instead to entanglement swapping.

As shown in Fig. 12, starting from the layer in Fig. 12(a), following the *data-qubit-swapping-based strategy*, in Fig. 12(b), we apply a remote SWAP between qubit 3 and qubit 4, and then, we can execute all remote CNOTs in parallel.

Moving from the topology illustrated in Fig. 7 to the one in Fig. 13 , we can devise a different strategy to execute swaps by exploiting the augmented connectivity, described in Algorithm 2. Specifically, to perform a SWAP between QPU Q_x and Q_y, our compiler uses one communication qubit at each QPU as a buffer memory and exploits quantum teleportation to move the state of a data qubit from Q_x to Q_y and *vice versa*, in parallel. After the two parallel teleportations, we can execute two parallel local SWAPs between communication qubits and data qubits at each QPU to effectively achieve

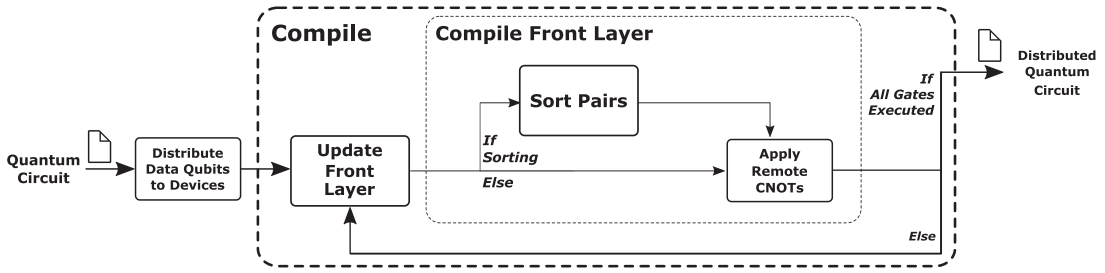


FIG. 11. Workflow of the proposed compiler.

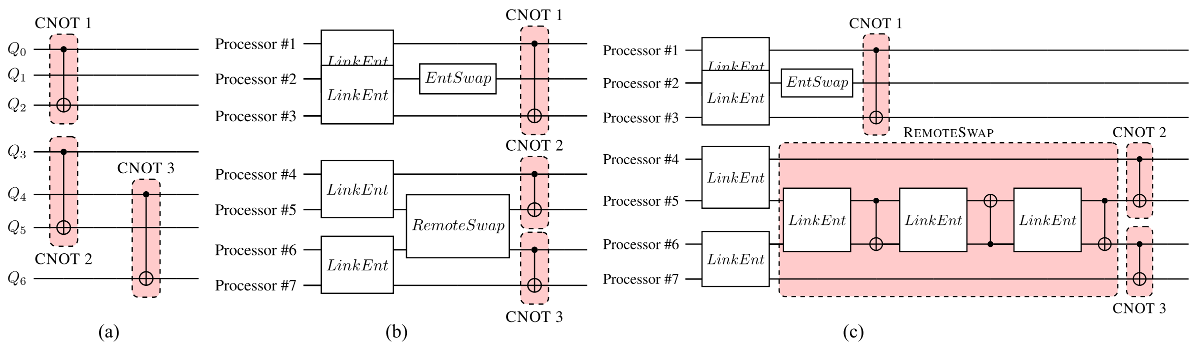


FIG. 12. Compilation of a layer with three parallel CNOTs using the sorting strategy. (a) Layer with three parallel CNOTs. (b) Layer distributed with the Sort strategy. (c) Distributed layer after decomposing the Remote SWAP.

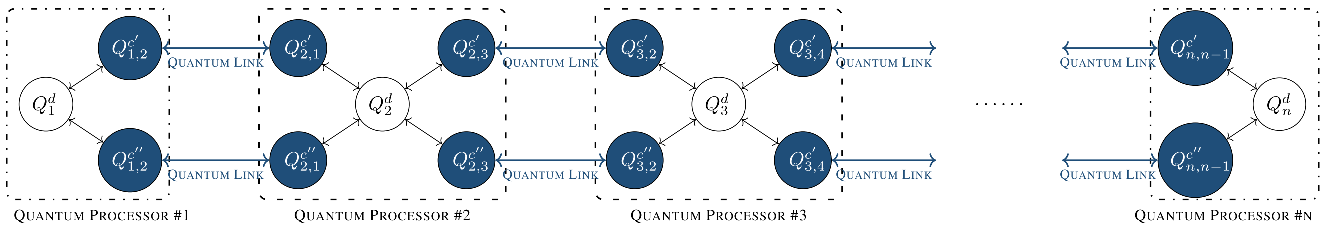


FIG. 13. Improving over the worst-case scenario topology. Only one data qubit is available at each quantum processor, but neighboring quantum processors are connected with two quantum links, realized by two EPR pairs.

data qubit swapping between Q_x and Q_y. This is clearly beneficial as it only requires one layer of link entanglement generation between the interested QPUs, unlike the previous scenario where we needed three layers of link entanglement generation to perform three remote CNOTs.

The difference between *data-qubit swapping* and *entanglement swapping* lies in the fact that, if the subsequent front layers are similar (in terms of CNOT interaction pattern) to the one just compiled, not much data swapping and very little entanglement swapping (given that the front layers involve most of the qubits) will be necessary to compile those layers.

Regarding the computational complexity, the compiler (whose behavior is summarized also by Algorithm 4 in the Appendix) reads the circuit from left to right while updating the front layer, so its computational complexity is $O(d)$, where d is the depth of the circuit, i.e., the number of layers. Algorithm 2 loops through all necessary swaps found by Algorithm 3, and applies remote CNOTs. As we must take into account for possible *entanglement swapping* operations between QPUs, applying a remote CNOT has a computational complexity of $O(n)$, with n being the number of QPUs. Given that we need at most $n/4$ swaps, the computational complexity of Algorithm 2 turns out to be $O(n^2)$. The overall computational complexity of distributing a circuit is, therefore, $O(dn^2)$.

B. COMPILING OVERHEAD VALIDATION

We validate the theoretical upper bounds (derived in Section IV-B) on the number of layers that result from compiling a layer of remote CNOTs, considering an extensive set of medium-size quantum circuits (the largest ones requiring 16 qubits, with the exception of a GHZ circuit, used to produce Greenberger–Horne–Zeilinger states, and two random circuit with 20 qubits). Specifically, we consider quantum circuits that are publicly available and widely adopted for testing quantum compilers [25], [26],[11] plus a few quantum chemistry circuits for the implementation of the unitary

[11]https://github.com/deeptechlabs/quantum_compiler_optim/tree/master/examples

Algorithm 3: SortPairs.
Input: a Vector v representing CNOT interactions between qubits pairs, ex. [1 2 2 3 1 3]
Output: the *swaps* to Perform.

```
// f.e.o. stands for "first element of"
if length($v$) mod4 $\neq 0$ then // length(v) must be multiple of 4
    add $\{-1, -1\}$ to $v$ // Add a dummy pair at the end
end if
$swaps \leftarrow \emptyset$ // List of SWAPs to perform
$n \leftarrow$ length($v$)
$mask1 \leftarrow \{0, \ldots, \frac{n}{2} - 1\}$
$mask2 \leftarrow \{\frac{n}{2}, \ldots, n - 1\}$
$cycle \leftarrow 0$
while $cycle < n/4$ do
    $w \leftarrow$ indexes of unique values in $v$
    $index\_last \leftarrow \{0, \ldots, |v[mask1]|\} \setminus w$
    if $index\_last = \emptyset$ then
        swap $v[mask1[end]]$ and
            f.e.o. s.t. $v[mask2] \neq v[mask1[end]]$
        update $swaps$
    end if
    $v, mask1, swaps \leftarrow$ swap($v$, $mask1$, $swaps$)
    $v, mask2, swaps \leftarrow$ swap($v$, $mask2$, $swaps$)
end while
return $swaps$
function SWAP($v, mask, swaps$)
    $w \leftarrow$ indexes of unique values in $v$
    $index\_last \leftarrow$ f.e.o. $\{0, \ldots, |v[mask]|\} \setminus w$
    $index\_first \leftarrow$ f.e.o.$v[mask]$ s.t. $= v[mask[index\_last]]$
    if $index\_last$ is odd then
        $index\_swap \leftarrow index\_last - 1$
    else
        $index\_swap \leftarrow index\_last + 1$
    end if
    swap $v[mask[index\_swap]]$ and $v[mask[index\_first]]$
    update $swaps$
    remove $mask[index\_swap]$ and $mask[index\_last]$ from $mask$
    return $v, mask, swaps$
end Function
```

quantum coupled cluster [48], [49] and the RYRZ heuristic [50] wavefunction Ansätze.

More into details, Table 1 reports a sample of the results that have been collected by compiling the circuits with both *entanglement-swapping-based* and *data-qubit-swapping-based* strategies. Within the table, the first column shows the name of the circuit and the second column shows the number n of logical qubits within the circuit. The third column shows the number of CNOT layers in the original circuit—i.e., the uncompiled circuit. The fourth and seventh columns show the theoretical upper bound of the depth of the compiled CNOT layers—computed in agreement with (2) and (4), respectively—whereas the fifth and eighth column show the depth of the compiled CNOT layers. For computing the upper bound values and collecting the experimental results, we set the parameters c_{le}, c_{bsm}, and c_{cx} in (3), (6), and (7) as unit factors, thus obtaining $d_{es} = 3$, $d_{qs} = 9$, and $d'_{qs} = 2$.

Table 1 clearly shows that the upper bounds on the number of layers that result from compiling the layers of remote CNOTs are widely respected and hold for all the considered examples. Indeed, by comparing the actual depth with the theoretical one, the overestimation of the bounds given in Section IV-B becomes evident. The rationale for this lays in the number of CNOTs in each layer, which are usually significantly lower than what assumed. For instance, let us consider the *GHZ* circuits, where each layer contains a single CNOT, and hence, the derived bounds—by assuming $n/2$ CNOTs in each layer—overestimate the depth. Nevertheless, the discrepancy can be easily fixed by substituting the $n/2$ factor with the actual estimation on the average number of CNOTs in each layer with no loss of generality.

C. EXPERIMENTAL RESULTS FOR THE WORST-CASE TOPOLOGY

We compared our compiler with the one proposed by Andrés-Martínez and Heunen [42], in respect of which we were able to set each QPU memory to one data qubit but we could not impose any limitation over the number of communication qubits per QPU nor the topology of the quantum network, which is always assumed to be an hypercube. Such a topology is shown in Fig. 14(b), where one can clearly see the connectivity disparity, compared to the worst-case topology illustrated in Fig. 14(a). In Figs. 15–17, the results of the comparative evaluation are plotted. For a better readability of the figures, we omit data related to a 20-qubit random circuit that presents values far greater than the rest of the dataset, for all compiling strategies including the state of the art (such data have been included in Table 1).

Fig. 15 shows the number of link generation layers, i.e., the number of layers in the distributed circuit that comprise only Bell state generation and distribution between QPUs. It is clear that our compiler requires less layers of link generation for almost every tested circuit. Having fewer layers of link generation reduces the time that data qubits have to spend idle, i.e., possibly affected by decoherence, while waiting to be able to perform remote operations. Fig. 15(c) also shows that choosing the *data-qubit-swapping-based strategy* against the *entanglement-swapping-based strategy* is generally the best choice.

Regarding the depth of the distributed circuits, illustrated in Fig. 16, we can see that for some circuits, our compiler clearly outperforms Andrés-Martínez's one. Fig. 16(c) confirms that the *data-qubit-swapping-based strategy* is better than the *entanglement-swapping-based* one.

TABLE 1. **Validation of the theoretical upper bounds derived in Section IV-B against a heterogeneous set of quantum circuits each circuit is characterized by a number of qubits *n* and a number of CNOT layers, i.e., layers that comprise only CNOT gates. For each compiling strategy, there is a theoretical upper bound on the number of layers that are necessary to realize the remote CNOTs and an actual number of layers resulting from the compilation process. The ratio between the latter and the former is also reported**

circuit name	# qubits n	# CNOT layers	Entanglement Swap			Data-Qubit Swap		
			# CNOT layers $\times \frac{n}{2} d_{es}$	compiled # layers	compiled/theoretical	# CNOT layers $\times (\frac{n}{4} d_{qs} + d'_{qs})$	compiled # layers	compiled/theoretical
4gt12-v1_89	16	88	2112	212	0.10	3344	233	0.07
4gt4-v0_73	16	160	3840	372	0.10	6080	399	0.07
4mod7-v1_96	16	65	1560	151	0.10	2470	155	0.06
9symml_195	16	12849	308376	34721	0.11	488262	32809	0.07
adder	10	55	825	144	0.17	1100	187	0.17
alu-v2_31	16	172	4128	459	0.11	6536	436	0.07
ghz_20	20	19	570	56	0.10	893	56	0.06
ghz_4	4	3	18	8	0.44	33	8	0.24
H2_RYRZ	4	25	150	66	0.44	275	69	0.25
H2_UCCSD	4	52	312	72	0.23	572	72	0.13
H2O_RYRZ	14	125	2625	971	0.37	3625	2040	0.56
H2O_UCCSD	14	12937	271677	16017	0.06	375173	16017	0.04
ising_model_16	16	20	480	31	0.06	760	31	0.04
life_238	16	8356	200544	22391	0.11	317528	21073	0.07
LiH_RYRZ	12	105	1890	711	0.38	3045	1547	0.51
LiH_UCCSD	12	7264	130752	9216	0.07	210656	9216	0.04
one-two-three-v2_100	16	29	696	76	0.11	1102	69	0.06
Random_20q_RYRZ	20	185	5550	1991	0.36	8695	4113	0.47
Random_20q_UCCSD	20	110497	3314910	130197	0.04	5193359	130197	0.03
random1_n5_d5	5	15	90	69	0.77	165	53	0.32
random2_n16_d16	16	48	1152	666	0.58	1824	588	0.32
rd53_138	16	42	1008	116	0.12	1596	100	0.06
root_255	16	5965	143160	16699	0.12	226670	15973	0.07
sqn_258	16	3719	89256	9713	0.11	141322	9210	0.07
sym9_146	16	91	2184	277	0.13	3458	254	0.07

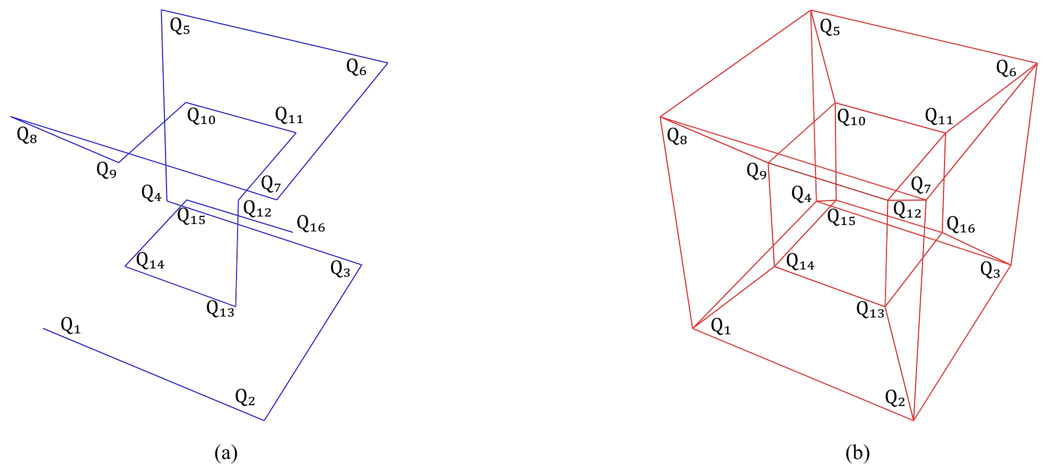


FIG. 14. **Comparison of the worst-case scenario LNN topology with the hypercube topology assumed by the compiler proposed by Andrés-Martínez and Heunen [42]. As shown in (a), for $n = 16$ QPUs, the longest path between two QPUs consists of $n - 1 = 15$ links, while with the hypercube topology in (b) is only about $\log_2 n = 3$ links. (a) LNN topology. (b) Hypercube topology.**

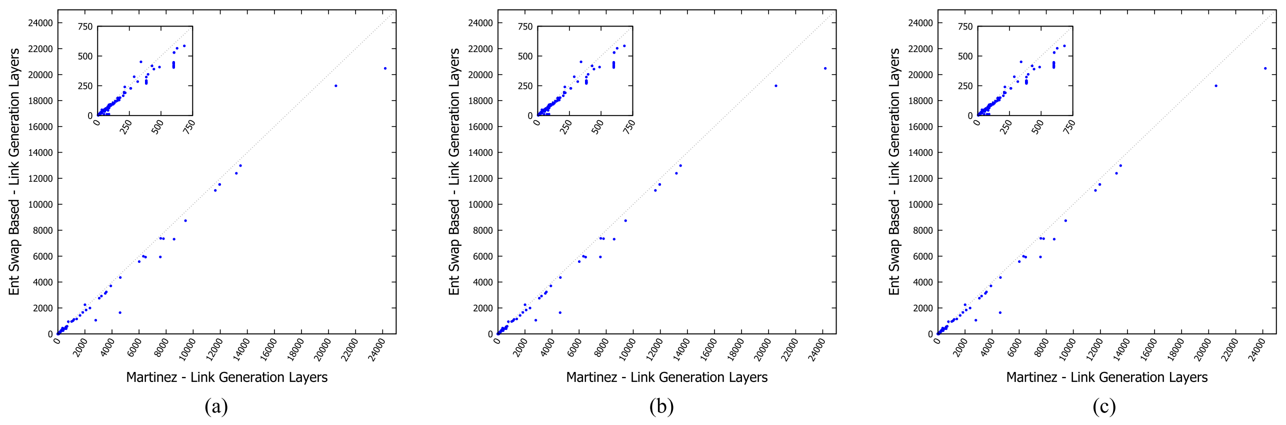


FIG. 15. **Comparing the *entanglement-swapping-based* and *data-qubit-swapping-based* strategies of our compiler with Andrés-Martínez's compiler [42], in terms of link entanglement generation layers. Our compiler distributes circuits on the worst-case topology, with only one link between neighboring QPUs (illustrated in Fig. 7), while Andrés-Martínez's one exploits a more favorable topology, i.e., the hypercube illustrated in Fig. 14(b). Both topologies are characterized by one data qubit per QPU.**

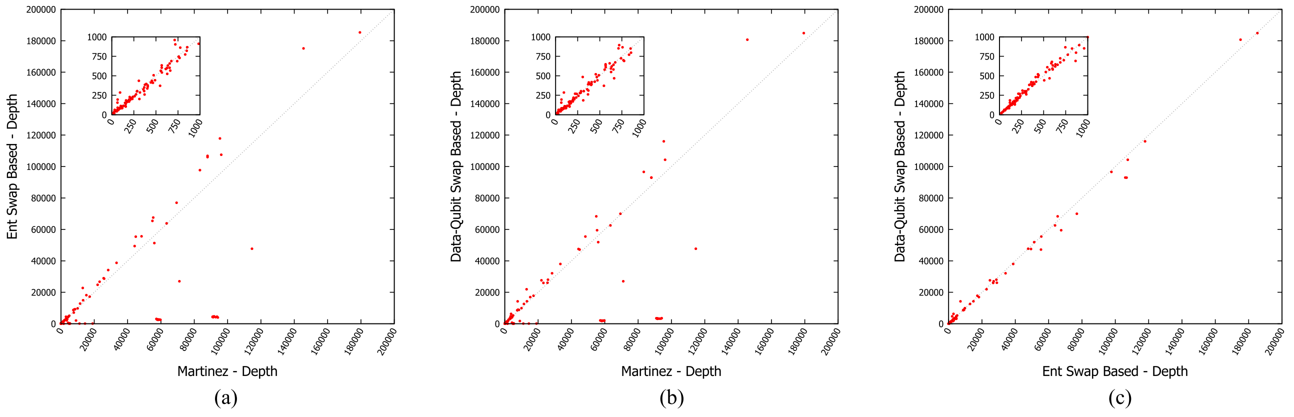


FIG. 16. **Comparing the *entanglement-swapping-based* and *data-qubit-swapping-based* strategies of our compiler with Andrés-Martínez's compiler [42], in terms of circuit depth. Our compiler distributes circuits on the worst-case topology, with only one link between neighboring QPUs (illustrated in Fig. 7), while Andrés-Martínez's one exploits a more favorable topology, i.e., the hypercube illustrated in Fig. 14(b). Both topologies are characterized by one data qubit per QPU.**

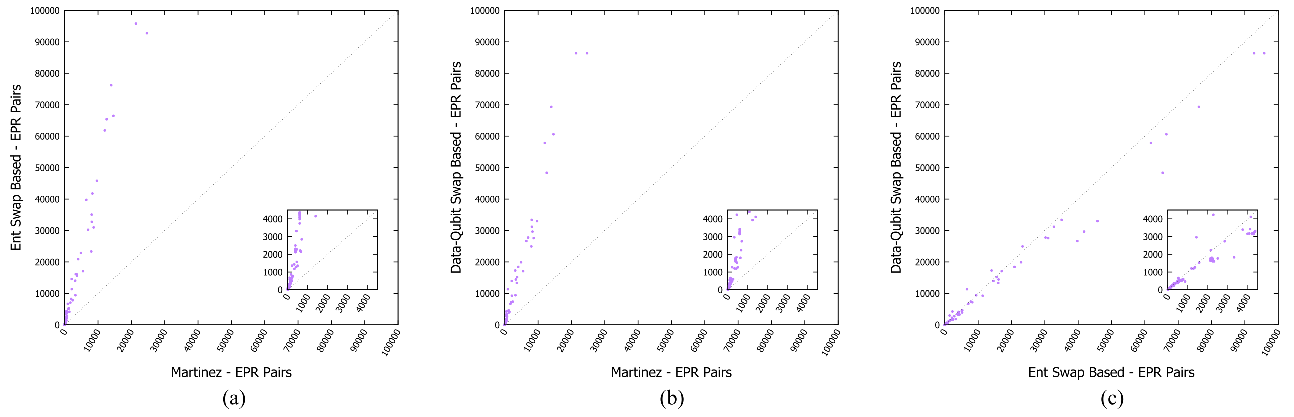


FIG. 17. **Comparing the *entanglement-swapping-based* and *data-qubit-swapping-based* strategies of our compiler with Andrés-Martínez's compiler [42], in terms of consumed EPR pairs. Our compiler distributes circuits on the worst-case topology, with only one link between neighboring QPUs (illustrated in Fig. 7), while Andrés-Martínez's one exploits a more favorable topology, i.e., the hypercube illustrated in Fig. 14(b). Both topologies are characterized by one data qubit per QPU.**

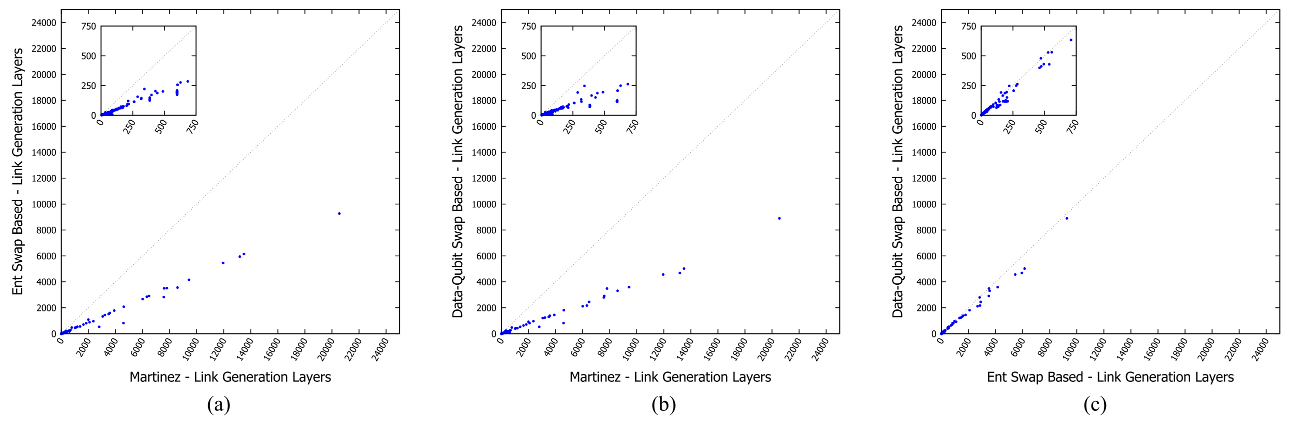


FIG. 18. **Comparing the *entanglement-swapping-based* and *data-qubit-swapping-based* strategies of our compiler with Andrés-Martínez's compiler [42] over link entanglement generation layers. Our compiler distributed circuits on the topology illustrated in Fig. 13, with two links between neighboring QPUs (illustrated in Fig. 7), while Andrés-Martínez's one exploits a more favorable topology, i.e., the hypercube illustrated in Fig. 14(b). Both topologies are characterized by one data qubit per QPU.**

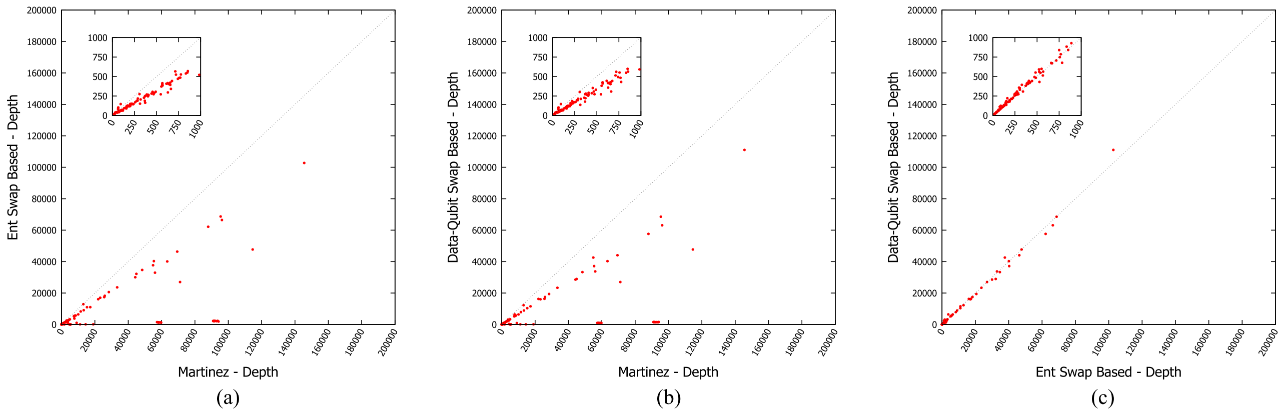


FIG. 19. **Comparing the *entanglement-swapping-based* and *data-qubit-swapping-based* strategies of our compiler with Andrés-Martínez's compiler [42], in terms of circuits' depth. Our compiler distributed circuits on the topology illustrated in Fig. 13, with two links between neighboring QPUs (illustrated in Fig. 7), while Andrés-Martínez's one exploits a more favorable topology, i.e., the hypercube illustrated in Fig. 14(b). Both topologies are characterized by one data qubit per QPU.**

With respect to the number of generated Bell states, depicted in Fig. 17, it can be observed that our compiler consumes a fair amount of Bell states compared to Andrés-Martínez's compiler. This was expected, and it is mostly due to the fact that Andrés-Martínez's compiler benefits from a hypercube network topology, as showed in Fig. 14(b). Using such a topology means that, in most cases, a link between two QPUs can be directly generated with just one Bell state, which is in direct contrast with the worst-case topology that we used, depicted in Fig. 14(a). In our linear topology, to generate a link between two nonneighboring QPUs, we need to perform entanglement swapping, generating, and consuming Bell states shared by all the others QPUs in between. Nevertheless, the time needed to generate one Bell state should be the same as to generate n Bell states in parallel, and with Fig. 15, we already showed that our compiler usually needs fewer layers of link generation.

It is worthwhile to note that with network topologies different from the considered one—namely, the worst-case topology where each CNOT must be mapped into a remote CNOT—new optimization challenges arise. As instance, whenever multiple data qubits are available at each (or some nodes), only a subset of CNOTs must be mapped into remote operations. Hence, the compiler should be able to optimize choices such as which subcircuit should be mapped to which node or which CNOT should be performed via communication qubits. Clearly, the optimal strategies—as well as the metrics to measure the optimality of a strategy—represents interesting open problems.

D. ADDITIONAL EXPERIMENTAL RESULTS

To show that the proposed compiling strategies can be applied to more complex topologies, we tested them on a slight variation of the worst-case scenario. This new topology is depicted in Fig. 13, where we doubled the number of EPR pairs per QPU; hence, we doubled the number of links between neighbor QPUs. As described in Section V-A, this setting enables our compiler to perform data-qubit swapping in a more efficient way and also greatly reduces the number of layers dedicated to the link entanglement generation, as we can generate and use double the number of links in parallel. Such a performance improvement is clearly shown in Fig. 18.

The same considerations apply to the depth of the compiled circuits, illustrated in Fig. 19, where we can observe an appreciable improvement against the worst-case scenario and the state-of-the-art compiler. As for the number of generated EPR pairs rendered in Fig. 20, aside from an imperceptible difference with the worst-case scenario, the advantage of an hypercube topology is still evident.

VI. CONCLUSION

In this article, we have discussed the main challenges arising with compiler design for distributed quantum computing. Then, we analytically derived an upper bound of the overhead induced by quantum compilation for distributed quantum computing. The derived bound accounts for the overhead induced by the underlying computing architecture as well as the additional overhead induced by the suboptimal quantum compiler. To this aim, we designed a quantum compiler with three key features: 1) general-purpose, namely, requiring no particular assumptions on the quantum circuits to be compiled; 2) efficient, namely, exhibiting a polynomial-time computational complexity so that it can successfully compile medium-to-large circuits of practical value; and 3) effective, being the total circuit depth overhead induced by the quantum circuit compilation always upper-bounded by a factor that grows linearly with the number of logical qubits of the original quantum circuit. We validated the theoretical upper bound against an extensive set of medium-size quantum circuits of practical interest, and we confirmed the validity of the compiler design through an extensive performance analysis.

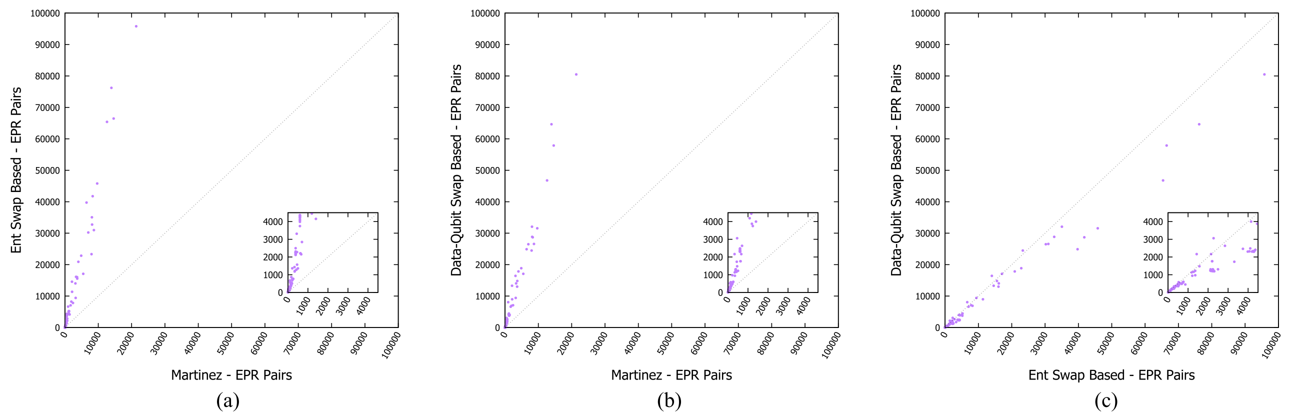


FIG. 20. Comparing the *entanglement-swapping-based* and *data-qubit-swapping-based* strategies of our compiler with Andrés-Martínez's compiler [42] over consumed EPR pairs. Our compiler distributed circuits on the topology illustrated in Fig. 13, with two links between neighboring QPUs (illustrated in Fig. 7), while Andrés-Martínez's one exploits a more favorable topology, i.e., the hypercube illustrated in Fig. 14(b). Both topologies are characterized by one data qubit per QPU.

```
Algorithm 4: COMPILE.
Input: the circuit to be Compiled
Output: the Compiled Circuit.
    $\mathcal{G} \leftarrow$ all gates from circuit
    $\mathcal{E} \leftarrow \emptyset$ // executed gates
    create new_circuit as an empty circuit
    while $\mathcal{G} \neq \emptyset$ do
      $\mathcal{G}, \mathcal{F}, \mathcal{E} \leftarrow$ UPDATEFRONTLAYER($\mathcal{G}, \mathcal{E}$) // $\mathcal{F}$ is the front layer
      if $\mathcal{F} \neq \emptyset$ then
        $\mathcal{E} \leftarrow$ COMPILEFRONTLAYER($\mathcal{F}, \mathcal{E}$)
      end if
    end while
    for all gate $\in \mathcal{E}$ do :
      add gate to new_circuit
    end for
    return new_circuit
```

APPENDIX

Here, we present a pseudocode description of the whole compilation process illustrated in Fig. 11 and summarized by Algorithm 4. The front layer is iteratively updated with Algorithm 5 and compiled by Algorithm 2. This is done until no more front layers can be computed, meaning that all gates have been mapped and the compilation process has finished.

ACKNOWLEDGMENT

This article benefited from the High Performance Computing facility of the University of Parma, Parma, Italy. The authors would like to thank Pablo Andrés-Martínez for answering questions about his related work.

```
Algorithm 5: UPDATEFRONTLAYER.
Input: $\mathcal{G}$ Gates to be Executed, $\mathcal{E}$ Executed Gates Until Now
Output: Updated $\mathcal{G}$, Updated $\mathcal{E}$, New Front Layer $\mathcal{F}$.
    $\mathcal{A} \leftarrow \emptyset$ // allocated qubits
    $\mathcal{F} \leftarrow \emptyset$ // new front layer
    $\mathcal{R} \leftarrow \emptyset$ // gates to remove
    width $\leftarrow$ total number of data qubits available
    for all gate $\in \mathcal{G}$ do
      if $|\mathcal{A}| = width$ then
        break
      end if
      if $\mathcal{A} \cap gate.qubits = \emptyset$ then
        if gate is a one-qubit gate then
          add gate to $\mathcal{E}$
        else
          add gate to $\mathcal{F}$
          add gate to $\mathcal{E}$
          add gate.qubits to $\mathcal{A}$
        end if
        add gate to $\mathcal{R}$
      else:
        add gate.qubits to $\mathcal{A}$
      end if
    end for
    for all gate $\in \mathcal{R}$ do
      remove gate from $\mathcal{G}$
    end for
    return $\mathcal{G}, \mathcal{F}, \mathcal{E}$
```

REFERENCES

[1] A. S. Cacciapuoti, M. Caleffi, F. Tafuri, F. S. Cataliotti, S. Gherardini, and G. Bianchi, "Quantum Internet: Networking challenges in distributed quantum computing," *IEEE Netw.*, vol. 34, no. 1, pp. 137–143, Jan./Feb. 2020, doi: 10.1109/MNET.001.1900092.

[2] M. Caleffi, A. S. Cacciapuoti, and G. Bianchi, "Quantum Internet: From communication to distributed computing!" in *Proc. 5th ACM Int. Conf. Nanoscale Comput. Commun.*, 2018, Art. no. 3, doi: 10.1145/3233188.3233224.

[3] M. Caleffi, D. Chandra, D. Cuomo, S. Hassanpour, and A. S. Cacciapuoti, "The rise of the quantum Internet," *Computer*, vol. 53, no. 6, pp. 67–72, 2020, doi: 10.1109/MC.2020.2984871.

[4] R. V. Meter and S. J. Devitt, "The path to scalable distributed quantum computing," *Computer*, vol. 49, no. 9, pp. 31–42, Sep. 2016, doi: 10.1109/MC.2016.291.

[5] S. Pirandola and S. L. Braunstein, "Physics: Unite to build a quantum internet," *Nature*, vol. 532, no. 7598, pp. 169–171, Apr. 2016, doi: 10.1038/532169a.

[6] E. Gibney, "Chinese satellite is one giant step for the quantum Internet," *Nature*, vol. 535, no. 7613, pp. 478–479, Jul. 2016, doi: 10.1038/535478a.

[7] W. D, R. Lamprecht, and S. Heusler, "Towards a quantum Internet," *Eur. J. Phys.*, vol. 38, no. 4, May 2017, Art. no. 043001, doi: 10.1088/1361-6404/aa6df7.

[8] C. Simon, "Towards a global quantum network," *Nat. Photon.*, vol. 11, no. 11, pp. 678–680, 2017, doi: 10.1038/s41566-017-0032-0.

[9] S. Wehner, D. Elkouss, and R. Hanson, "Quantum internet: A vision for the road ahead," *Science*, vol. 362, no. 6412, 2018, Art. no. eaam9288, doi: 10.1126/science.aam9288.

[10] M. Zomorodi-Moghadam, M. Houshmand, and M. Houshmand, "Optimizing teleportation cost in distributed quantum circuits," *Int. J. Theor. Phys.*, vol. 57, pp. 848–861, 2018, doi: 10.1007/s10773-017-3618-x.

[11] L. Gyongyosi and S. Imre, "Entanglement concentration service for the quantum Internet," *Quantum Inf. Process.*, vol. 19, no. 8, 2020, Art. no. 221, doi: 10.1007/s11128-020-02716-3.

[12] L. Gyongyosi and S. Imre, "Routing space exploration for scalable routing in the quantum Internet," *Sci. Rep.*, vol. 10, no. 1, 2020, Art. no. 11874, doi: 10.1038/s41598-020-68354-y.

[13] D. Cuomo, M. Caleffi, and A. S. Cacciapuoti, "Towards a distributed quantum computing ecosystem," *IET Quantum Commun.*, vol. 1, pp. 3–8, Jul. 2020, doi: 10.1049/iet-qtc.2020.0002.

[14] A. Botea, A. Kishimoto, and R. Marinescu, "On the complexity of quantum circuit compilation," in *Proc. Symp. Combinatorial Search*, 2018, pp. 138–142.

[15] E. G. Rieffel and W. H. Polak, *Quantum Computing: A Gentle Introduction*, Cambridge, MA, USA: MIT Press, 2011, doi: 10.5555/1973124.

[16] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge, U.K.: Cambridge Univ. Press, 2010, doi: 10.1017/CBO9780511976667.

[17] D. Deutsch, "Quantum theory, the Church-Turing principle and the universal quantum computer," *Proc. Roy. Soc. London A, Math., Phys. Eng. Sci.*, vol. 400, pp. 97–117, 1985, doi: 10.1098/rspa.1985.0070.

[18] S. Boixo, "Characterizing quantum supremacy in near-term devices," *Nat. Phys.*, vol. 14, no. 6, pp. 595–600, 2018, doi: 10.1038/s41567-018-0124-x.

[19] A. Kandala, K. Temme, A. D. Crcoles, A. Mezzacapo, J. M. Chow, and J. M. Gambetta, "Error mitigation extends the computational reach of a noisy quantum processor," *Nature*, vol. 567, no. 7749, pp. 491–495, 2019, doi: 10.1038/s41586-019-1040-7.

[20] L. Gyongyosi and S. Imre, "Circuit depth reduction for gate-model quantum computers," *Sci. Rep.*, vol. 10, no. 1, 2020, Art. no. 11229, doi: 10.1038/s41598-020-67014-5.

[21] A. S. Cacciapuoti, M. Caleffi, R. Van Meter, and L. Hanzo, "When entanglement meets classical communications: Quantum teleportation for the quantum internet," *IEEE Trans. Commun.*, vol. 68, no. 6, pp. 3808–3833, Jun. 2020, doi: 10.1109/TCOMM.2020.2978071.

[22] IBM, IBM quantum systems, 2020. [Online]. Available: https://quantum-computing.ibm.com/docs/cloud/backends/systems/system-backends-systems-available

[23] J. C. Garcia-Escartin and P. Chamorro-Posada, "Equivalent quantum circuits," 2011, *arXiv:1110.2998*.

[24] A. D. Crcoles *et al.*, "Challenges and opportunities of near-term quantum computing systems," *Proc. IEEE*, vol. 108, no. 8, pp. 1338–1352, Aug. 2020, doi: 10.1109/JPROC.2019.2954005.

[25] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the IBM QX architectures," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 38, no. 7, pp. 1226–1236, Jul. 2019, doi: 10.1109/TCAD.2018.2846658.

[26] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for NISQ-era quantum devices," in *Proc. Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2019, pp. 1001–1014, doi: 10.1145/3297858.3304023.

[27] *Qiskit: An Open-Source Framework for Quantum Computing*, IBM, Armonk, NY, USA, 2019.

[28] D. Ferrari and M. Amoretti, "Efficient and effective quantum compiling for entanglement-based machine learning on IBM Q devices," *Int. J. Quantum Inf.*, vol. 16, no. 08, 2018, Art. no. 1840006, doi: 10.1142/S0219749918400063.

[29] L. Cincio, Y. Subaşı, A. T. Sornborger, and P. J. Coles, "Learning the quantum algorithm for state overlap," *New J. Phys.*, vol. 20, no. 11, Nov. 2018, Art. no. 113022, doi: 10.1088/1367-2630/aae94a.

[30] W. Kozlowski, S. Wehner, R. Van Meter, B. Rijsman, A. S. Cacciapuoti, and M. Caleffi, "Architectural principles for a quantum internet," Internet Engineering Task Force, Internet-Draft draft-irtf-qirg-principles-03, Mar. 2020.

[31] N. M. Linke *et al.*, "Experimental comparison of two quantum computing architectures," *Proc. Nat. Acad. Sci.*, vol. 114, no. 13, pp. 3305–3310, 2017, doi: 10.1073/pnas.1618020114.

[32] R. Van Meter and K. M. Itoh, "Fast quantum modular exponentiation," *Phys. Rev. A*, vol. 71, May 2005, Art. no. 052320, doi: 10.1103/PhysRevA.71.052320.

[33] A. G. Fowler, S. J. Devitt, and L. C. L. Hollenberg, "Implementation of SHOR's algorithm on a linear nearest neighbor qubit array," *Quantum Inf. Comput.*, vol. 4, pp. 237–251, 2004, doi: 10.5555/2011827.2011828.

[34] Y. Takahashi, N. Kunihiro, and K. Ohta, "The quantum Fourier transform on a linear nearest neighbor architecture," *Quantum Inf. Comp.*, vol. 7, pp. 383–391, 2007, doi: 10.5555/2011725.2011732.

[35] R. Van Meter, "Communications topology and distribution of the quantum Fourier transform," in *Proc. Quantum Inf. Technol. Symp.*, 2004.

[36] S. A. Kutin, "Shor's algorithm on a nearest-neighbor machine," 2007, *arXiv:quant-ph/0609001*.

[37] R. Van Meter, W. J. Munro, K. Nemoto, and K. M. Itoh, "Arithmetic on a distributed-memory quantum multicomputer," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 3, no. 4, Jan. 2008, Art. no. 2, doi: 10.1145/1324177.1324179.

[38] B.-S. Choi and R. Van Meter, "On the effect of quantum interaction distance on quantum addition circuits," *J. Emerg. Technol. Comput. Syst.*, vol. 7, no. 3, 2011, Art. no. 11, doi: 10.1145/2000502.2000504.

[39] R. Beals *et al.*, "Efficient distributed quantum computing," *Proc. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 469, no. 2153, 2013,Art. no. 20120686, doi: 10.1098/rspa.2012.0686.

[40] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: USA: MIT Press, 2001.

[41] Z. Davarzani, M. Zomorodi-Moghadam, M. Houshmand, and M. Nouri-Baygi, "A dynamic programming approach for distributing quantum circuits by bipartite graphs," *Quantum Inf. Process.*, vol. 19, 2020, Art. no. 360, doi: 10.1007/s11128-020-02871-7.

[42] P. Andrés-Martínez and C. Heunen, "Automated distribution of quantum circuits via hypergraph partitioning," *Phys. Rev. A*, vol. 100, Sep. 2019, Art. no. 032308, doi: 10.1103/PhysRevA.100.032308.

[43] Y. Akhremtsev, T. Heuer, P. Sanders, and S. Schlag, "Engineering a direct *k*-way hypergraph partitioning algorithm," in *Proc. Meeting Algorithm Eng. Exp.*, 2017, pp. 28–42, doi: 10.1137/1.9781611974768.3.

[44] M. Caleffi, "Optimal routing for quantum networks," *IEEE Access*, vol. 5, pp. 22 299–22 312, 2017, doi: 10.1109/ACCESS.2017.2763325.

[45] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching*. vol. 3, 2nd ed. Boston, MA, USA: Addison Wesley Longman, 1998.

[46] T. Ono, R. Okamoto, M. Tanida, H. F. Hofmann, and S. Takeuchi, "Implementation of a quantum controlled-SWAP gate with photonic circuits," *Sci. Rep.*, vol. 7, no. 1, 2017, Art. no. 45353, doi: 10.1038/srep45353.

[47] N. Schuch and J. Siewert, "Natural two-qubit gate for quantum computation using the XY interaction," *Phys. Rev.*, vol. 67, Mar. 2003, Art. no. 032301, doi: 10.1103/PhysRevA.67.032301.

[48] A. Peruzzo *et al.*, "A variational eigenvalue solver on a photonic quantum processor," *Nat. Commun.*, vol. 5, 2014, Art. no. 4213, doi: 10.1038/ncomms5213.

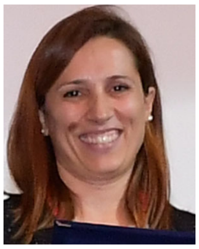
[49] P. K. Barkoutsos *et al.*, "Quantum algorithms for electronic structure calculations: Particle-hole Hamiltonian and optimized wave-function expansions," *Phys. Rev. A*, vol. 98, Aug. 2018, Art. no. 022322, doi: 10.1103/PhysRevA.98.022322.

[50] A. Kandala *et al.*, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *Nature*, vol. 549, no. 7671, pp. 242–246, Sep. 2017, doi: 10.1038/nature23879.

Davide Ferrari (Graduate Student Member, IEEE) received the B.Sc. degree in computer engineering from the Polytechnic of Milan, Milan, Italy, in 2016, and the M.Sc. degree in computer engineering from the University of Parma, Parma, Italy, in 2019, where he is currently working toward the Ph.D. degree with the Department of Engineering and Architecture.

He has been a Research Scholar with the Future Technology Laboratory, University of Parma, working on the design of efficient algorithms for quantum compiling. He is involved in the Quantum Information Science research initiative with the University of Parma, where he is a member of the Quantum Software research unit. His research interests include efficient quantum compiling for quantum simulations and quantum machine learning.

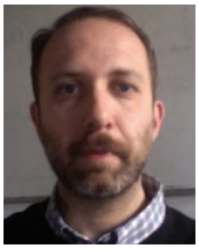
Mr. Ferrari won the IBM Quantum Awards Circuit Optimization Developer Challenge, in 2020.



Angela Sara Cacciapuoti (Senior Member, IEEE) received the “Laurea” (integrated B.S./M.S.) (*summa cum laude)* degree in telecommunications engineering and the Ph.D. degree in electronic and telecommunications engineering from the University of Naples Federico II, Naples, Italy, in 2005 and 2009, respectively.

She is currently an Associate Professor with the University of Naples Federico II. She was a Visiting Researcher with the Georgia Institute of Technology, Atlanta, GA, USA, and the Universitat Politecnica de Catalunya, Barcelona, Spain. Since July 2018, she has held the national habilitation as a Full Professor in Telecommunications Engineering. Her work has appeared in first-tier IEEE journals. Her current research interests include quantum communications, quantum networks, and quantum information processing.

Dr. Cacciapuoti is an Area Editor for the IEEE Communications Letters and an Editor/Associate Editor for the IEEE Transactions on Communications, the IEEE Transactions on Wireless Communications, the IEEE Transaction on Quantum Engineering, the IEEE Network, and the IEEE Open Journal of the Communications Society. She has received various awards. She was a recipient of the 2017 Exemplary Editor Award of the IEEE Communications Letters. In 2016, she was an appointed member of the IEEE Communications Society (ComSoc) Young Professionals Standing Committee. From 2017 to 2018, she was the Award Co-Chair of the N2Women Board. From 2017 to 2020, she was the Treasurer of the IEEE Women in Engineering Affinity Group of the IEEE Italy Section. In 2018, she was appointed as the Publicity Chair of the IEEE ComSoc Women in Communications Engineering (WICE) Standing Committee. Since 2020, she has been the Vice-Chair of WICE.



Michele Amoretti (Senior Member, IEEE) received the Ph.D. degree in information technologies from the University of Parma, Parma, Italy, in 2006.

He is currently an Associate Professor of Computer Engineering with the University of Parma. In 2013, he was a Visiting Researcher with LIG Lab, Grenoble, France. He authored or coauthored more than 100 research papers in refereed international journals, conference proceedings, and books. He is involved in the Quantum Information Science research and teaching initiative with the University of Parma, where he leads the Quantum Software research unit. He is the CINI Consortium delegate in the CEN-CENELEC Focus Group on Quantum Technologies. His current research interests include high-performance computing, quantum computing, and the Internet of Things.

Dr. Amoretti is an Associate Editor for the IEEE Transactions on Quantum Engineering and the *International Journal of Distributed Sensor Networks*.

Marcello Caleffi (Senior Member, IEEE) received the M.S. degree (*summa cum laude*) in computer science engineering from the University of Lecce, Lecce, Italy, in 2005, and the Ph.D. degree in electronic and telecommunications engineering from the University of Naples Federico II, Naples, Italy, in 2009.

He is currently an Associate Professor with the Department of Electrical Engineering and Information Technology, University of Naples Federico II. From 2010 to 2011, he was a Visiting Researcher with the Broadband Wireless Networking Laboratory, Georgia Institute of Technology, Atlanta, GA, USA. In 2011, he was also a Visiting Researcher with the NaNoNetworking Center in Catalunya, Universitat Politecnica de Catalunya, Barcelona, Spain. Since July 2018, he has held the Italian national habilitation as a Full Professor in Telecommunications Engineering. His work appeared in several premier IEEE transactions and journals.

Dr. Caleffi received multiple awards, including Best Strategy Award, Most Downloaded Article Awards, and Most Cited Article Awards. He is an Associate Technical Editor for the *IEEE Communications Magazine* and an Associate Editor for the IEEE Transactions on Quantum Engineering and the IEEE Communications Letters. He served as Chair, Technical Program Committee (TPC) Chair, Session Chair, and TPC Member for several premier IEEE Conferences. In 2017, he became a Distinguished Lecturer for the IEEE Computer Society. In December 2017, he has been elected Treasurer of the Joint IEEE Vehicular Technology Society/IEEE Communications Society Chapter Italy Section. In December 2018, he was appointed a Member of the IEEE New Initiatives Committee.