UNIVERSITÀ DI PARMA

UNIVERSITÀ DEGLI STUDI PARMA

DOTTORATO DI RICERCA IN
"TECNOLOGIE DELL'INFORMAZIONE"

CICLO XXXV

Compilation Strategies For Local And Distributed Quantum Computing

Coordinatore:
Chiar.mo Prof. Marco Locatelli

Tutore:
Chiar.mo Prof. Michele Amoretti

Dottorando: Davide Ferrari

Anni Accademici 2019/2020 - 2021/2022

UNIVERSITÀ DEGLI STUDI DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXXV Ciclo

Compilation Strategies For Local And Distributed Quantum Computing

Coordinatore:

Chiar.mo Prof. Marco Locatelli

Tutor:

Chiar.mo Prof. Michele Amoretti

Dottorando: *Davide Ferrari*

Anni Accademici 2019/2020 - 2021/2022

To my family
and friends

Contents

Introduction

The idea of Quantum Computing arose in 1982 during a speech by Richard Feynman about the difficulty of simulating quantum mechanical systems with classical computers [1]. Feynman suggested the simulation of these systems using quantum computers in other words, controlled quantum mechanical systems.

Since then, quantum computing and quantum information theory continued to advance, proving that universal quantum computers would be more powerful than Turing Machines [2]. Quantum computing will have a deep impact on a variety of fields, from quantum simulation in physics and chemistry [3], to machine learning [4], artificial intelligence [5] and cryptography [6, 7].

Current quantum computers are commonly defined as *noisy intermediate-scale quantum (NISQ)* [8] devices, being characterized by few hundreds of quantum bits (qubits) with non-uniform quality and constrained physical connectivity.

Quantum compilation is the problem of translating an input quantum circuit into the most efficient equivalent of itself [9], taking into account the characteristics of the device that will execute the computation. In general, the quantum compilation problem is NP-Hard [10, 11]. Compilation strategies are composed by sequential *passes* that perform placement, routing and optimization tasks. Several routing and optimization passes may be executed within a compilation strategy.

There exist several quantum algorithms known or expected to outperform classical algorithms for problems spanning different areas, including cryptography, search and optimization, simulation of quantum systems and learning [12]. For most practical applications, quantum algorithms require large quantum computing resources

– in terms of qubit number – much larger than those available with current NISQ processors. Hence, the growing demand for large-scale quantum computers is motivating research on *Distributed Quantum Computing* (*DQC*) architectures [13, 14] as a scalable approach for increasing the number of qubits available for computational tasks.

With the network and communications functionalities provided by the *Quantum Internet* [15, 16, 17, 18, 19, 20, 13, 14, 21, 22], remote quantum processing units (QPUs) can communicate and cooperate for executing computational tasks that each NISQ device cannot handle by itself. Accordingly, DQC requires a new generation of quantum compilers, for mapping any quantum algorithm to any DQC architecture.

In general, when moving from local to distributed quantum computing one faces several challenges. A first issue that arises with distributed quantum computing is whether a given algorithm – equivalently, a given quantum circuit – is natively suitable for distributed execution. More specifically, a *perfectly distributable* quantum algorithm is a quantum algorithm that can be split into autonomous parts that do not interact – or, at least, weakly interact – with each others. Unfortunately, this is rarely the case. Another challenge that one must face in DQC is related to circuit partitioning. To partition a monolithic quantum algorithm, a quantum compiler must be used to find the best breakdown, specifically, one that minimizes the number of gates that are applied to qubits stored at different devices.

In this Thesis we propose different strategies for the placement and routing of quantum circuits for local and distributed quantum computing. We also design and test a general-purpose quantum compilation framework for DQC that takes into account both network and devices constraints and characteristics. The Thesis is structured as follows. In Chapter 1 we revise the basics of quantum computation, introduce the notions of quantum networking and DQC, and present the state of the art in quantum compilation for local and distributed quantum computing. Then, in Chapter 2 we propose a general-purpose quantum compilation framework and explain the strategies devised for the placement and routing of quantum circuits, including a strategy to combine compilation for local and distributed quantum computing. Following, in Chapter 3 we deal with implementation details. Finally, in Chapter 4 we illustrate the

results obtained with the aforementioned quantum compilation strategies. We conclude the Thesis with a discussion of the obtained results and open problems.

Chapter 1

State of the Art

1.1 Background

In this section, we illustrate the main concepts of quantum computing and distributed quantum computing. In particular we focus on the technical aspects that are relevant in the contest of quantum compiling.

1.1.1 Quantum Computing

In 1982 Richard Feynman gave a speech about the difficulty of simulating quantum mechanical systems with classical computers [1]. The number of superposition states characterizing those systems is too large for being efficiently represented and processed by classical computers.

Therefore, Feynman suggested the implementation of quantum computers, *i.e.*, controlled quantum mechanical systems able to mimic the natural quantum mechanical systems. The simulation of these systems using quantum computers, whose initial state evolves under the same principles as the simulated system, is considered to be the "killer application" of quantum computing.

The no-cloning theorem was one of the earliest results of quantum computing [23, 24], forbidding the creation of identical copies of an arbitrary unknown quantum system state. While the concept of qubit was introduced by Stephen Wiesner [25] in

his paper about unforgeable quantum money, the term qubit was coined by Benjamin Schumacher in 1995 [26].

In 1985, David Deutsch [27] investigated the possibility to use the laws of physics to derive a stronger version of the Church-Turing thesis. He proposed the Quantum Turing Machine (QTM), i.e., a computational device that is capable of efficiently simulating any physical system. In what is now known as the Deutsch's algorithm, he constructed a simple one qubit example which suggests that quantum computers might actually exceed classical computers in terms of computational power.

In 1994, Peter Shor [2] proved that finding the prime factors of an integer and the discrete logarithm k that solves the equation $x^k = y$, where x and y belong to a finite group, could be efficiently solved on a quantum computer. These are important problems for cryptography and known to be in both *NP* and *coNP*. Shor's results indicate that quantum computers are more powerful than Turing Machines, even nondeterministic ones.

Qubit

It is common knowledge that the *bit* is the fundamental concept behind classical computation and classical information. *Quantum computation* and *quantum information* are based on a similar concept, the *quantum bit*, also known as *qubit* [25, 26, 28]. It is convenient to focus on qubits as mathematical objects, but the reader should keep in mind that they can be realized as actual multi-level physical systems such as the spin of the electron, the polarization of a photon, or an atom with a ground state and an excited state.

A qubit is the simplest quantum-mechanical system. Its state is defined as a vector in the complex Hilbert space, whose mathematical general form is

$$|\phi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1.1}$$

where

- $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ is the *ket notation* for the two

orthonormal vectors of the *computational basis* states

- $\alpha, \beta \in \mathbb{C}$ are called *probability amplitudes*, with $|\alpha|^2 + |\beta|^2 = 1$

Importantly, $|\phi\rangle$ is a *unit vector* with its norm being $\||\phi\rangle\| = 1$.

Until a *measurement* is made, the state of a qubit may be a *superposition* of basis states, and, if α and β are unknown, there is no way to know their values with a single measurement. The result of the measurement in the computational basis will be $|0\rangle$ or $|1\rangle$ with probability $|\alpha|^2$ or $|\beta|^2$ respectively.

Since $|\alpha|^2 + |\beta|^2 = 1$, Eq. (1.1) can be restated as

$$|\phi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle \tag{1.2}$$

where $\alpha = \cos\frac{\theta}{2}$ is a real number, but can always be made complex by multiplying $|\phi\rangle$ by an overall phase factor (which is unobservable). Here, φ is known as the relative phase of the quantum state and it is crucial in creating the so called interference patterns exploited by many quantum algorithms. From Eq. (1.2) comes the *Bloch sphere*, a three-dimensional representation of the state space of a qubit in spherical coordinates (Figure 1.1).

The above theory can be generalized to a composite system of n-qubits, associated to an Hilbert space of dimension 2^n, since the vector spaces of the constituent quantum systems are combined through the tensor product. The state of an n-qubit system can than be in a superposition of all the 2^n basis states:

$$|\phi\rangle = \sum_{k=0}^{2^n-1} \alpha_k |k\rangle, \tag{1.3}$$

with $\alpha_k \in \mathbb{C} : \sum_{k=0}^{2^n-1} |\alpha_k|^2 = 1$.

Not all n-qubit systems states can be written as the tensor product of their constituent sub-systems states. These states are referred to as *entangled* states and they represent the key ingredient in quantum computing [29]. One popular two-qubit entangled state, referred to as *Bell state* [30] or *EPR pair* [31], is given by:

$$\left|\Phi^+\right\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}. \tag{1.4}$$

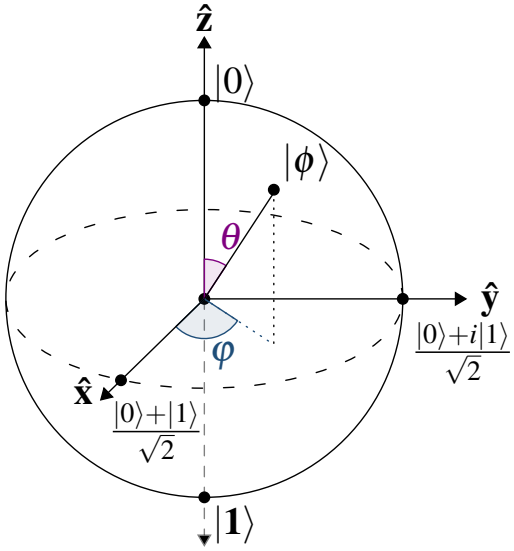


Figure 1.1: Bloch sphere, a geometrical representation of a qubit in spherical coordinates. A state $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$ is represented by a point on the sphere surface, with $\alpha = \cos\frac{\theta}{2}$ and $\beta = e^{i\varphi}\sin\frac{\theta}{2}$.

Quantum Circuits

Quantum systems evolve through *unitary transformations*. A unitary transformation is a complex matrix U such that:

$$UU^{\dagger} = U^{\dagger}U = I$$

with $U^{\dagger}$ being the transposed conjugate of U and I the identity matrix. A qubit state $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$ evolves to the state $|\phi\rangle'$ under the application of a 2×2 unitary matrix U such that:

$$|\phi\rangle' = U|\phi\rangle = \begin{pmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{pmatrix}\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = (u_{00}\alpha + u_{01}\beta)|0\rangle + (u_{10}\alpha + u_{11}\beta)|1\rangle . \tag{1.5}$$

By analogy with the classical computing model, operations on qubits such as the one described in Eq. (1.5) are called *gates*. Unlike classical gates, quantum gates are

reversible operations, reason why number of inputs and outputs are always equal. Table 1.1 shows the most common quantum gates.

Gate Name	Gate Matrix	Operation
Identity	$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	does not modify the quantum state
Pauli-X	$\mathtt{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	bit-flip
Pauli-Y	$\mathtt{Y} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	combined bit- and phase-flip
Pauli-Z	$\mathtt{Z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	phase-flip
Square-root-of-X	$\mathtt{SX} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1+i & 1-i \\ 1-i & 1+i \end{pmatrix}$	$\mathtt{SX}$ such that $\mathtt{X} = \mathtt{SX} \circ \mathtt{SX}$
Hadamard	$\mathtt{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	maps an element of the computational basis – either $\lvert 0 \rangle$ or $\lvert 1 \rangle$ – into an even superposition of the basis elements (and vice versa)
S	$\mathtt{S} = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$	$\pi/2$ phase shift
T	$\mathtt{T} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$	$\pi/4$ phase shift
Phase shift	$\mathtt{P}_\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$	θ phase shift
x-Rotation	$\mathtt{R}_x(\theta) = e^{-i\frac{\theta}{2}X} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}X$	rotation by θ along $\hat{x}$-axis of the Bloch sphere
y-Rotation	$\mathtt{R}_y(\theta) = e^{-i\frac{\theta}{2}Y} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}Y$	rotation by θ along $\hat{y}$-axis of the Bloch sphere
z-Rotation	$\mathtt{R}_z(\theta) = e^{-i\frac{\theta}{2}Z} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}Z$	rotation by θ along $\hat{z}$-axis of the Bloch sphere
Controlled-NOT	$\mathtt{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	bit-flips the second (target) qubit whenever the first (control) qubit is $\lvert 1 \rangle$

Table 1.1: Common quantum gates.

The *Pauli-X* gate, shortly denoted by *X*, is the quantum equivalent of the classical

NOT gate and switches the probability amplitudes of the qubit it acts on. It is actually a rotation by π radiants around the x axis of the Bloch sphere.

The *Hadamard* or H gate is a rotation by $\pi/2$ radiants around the y axis, followed by a reflection through the xy plane. When applied to a qubit in the computational basis state, it turns it into a superposition of the computational basis states. The H gate is self-inverse, *i.e.* it does not change the state of the qubit if applied twice as $H^2 = I$.

Among two-qubit gates, highly relevant are the *controlled* ones. The generic `Controlled-U` gate operates on two qubits, namely a *control qubit* (controlling the operation) and a *target qubit* (subjected to the operation). By denoting with $|\phi_c\rangle$ and $|\phi_t\rangle$ the control and target qubits respectively, the effect of the controlled U gate on the target qubit is the following:

$$\begin{cases} \mathtt{I}\,|\phi_t\rangle & \text{if } |\phi_c\rangle = |0\rangle \\ \mathtt{U}\,|\phi_t\rangle & \text{if } |\phi_c\rangle = |1\rangle\,. \end{cases} \tag{1.6}$$

The most famous example of controlled gate is represented by the `CNOT` gate, which is a `Controlled-U` gate where the `U` gate is a `Pauli-X` one.

Gates `H`, `S` and `CNOT` constitute the *Clifford group* [32], which can be simulated efficiently on a classical computer according to the Gottesman-Knill theorem [28]. The Clifford group is not universal, i.e., it cannot be used to describe any arbitrary quantum algorithm. However, it is sufficient to add the `T` gate to the Clifford group, and the resulting set is universal, yet it is not the only possible one. Indeed, each family of quantum computers – e.g., IBM Q one [33] – has its own specific universal gate set, which usually depends on the particulars of the underlying quantum hardware technology.

In the *quantum circuit* model of quantum computation, *quantum algorithms* are defined by sequentially interconnecting different quantum gates. In a quantum circuit, qubits are represented by horizontal lines and gates are elements placed on these lines, from left to right in the order they are executed. 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. 1.2 is composed of 9 layers and hence its depth is equal to 9. The number of gates within the circuit is denoted as *circuit size* and it is equal to 23 in Fig. 1.2.

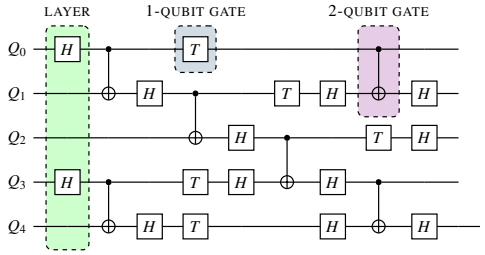


Figure 1.2: Example of a 5-qubit quantum circuit from [34], time flows from left to right. The first qubit undergoes through a single-qubit `H` gate followed by a two-qubit `CNOT` gate (represented by • and ⊕ symbols interconnected by a vertical line), a single-qubit `T` gate, and finally another two-qubit `CNOT` gate.

In Figure 1.3, we can see a simple quantum circuit that starts with two qubits in the $|0\rangle$ state, then applies an H gate to the first qubit and a CNOT gate, with the first qubit being the control and the second one being the target. The final state of the system is the Bell state $|\beta_{00}\rangle = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$. Another interesting example of quantum circuit is the one in Figure 1.4. Due to a *Pauli-X* gate, the state of the second qubit changes from $|0\rangle$ to $|1\rangle$. Then, a *SWAP* gate composed of three CNOTs is applied. The resulting state is $|10\rangle$ because the SWAP gate exchanges the states of the two qubits.

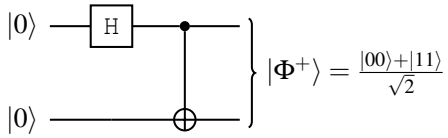


Figure 1.3: Quantum circuit to create a Bell pair.

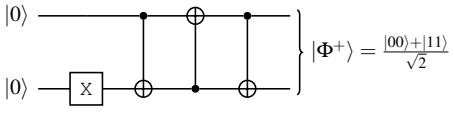


Figure 1.4: Quantum circuit with a SWAP.

It is worthwhile to note that, regardless the particulars of the adopted gate set,

deterministic cloning of quantum states is impossible. Specifically, there exists no quantum gate (or circuit) able to make a perfect copy of an arbitrary unknown quantum state. Conversely, if the state is known in advance – specifically, if we know that the state belongs to some orthonormal basis such as $\{|0\rangle, |1\rangle\}$ or $\{|+\rangle, |-\rangle\}$ – we can design a specific quantum gate to clone that state. This fundamental property is known as *no-cloning theorem*.

Another unconventional quantum phenomenon arises with the important operation constituted by *measurement*, through which information from a quantum state is extracted [35], as illustrated in Figure 1.5. In fact, according to the quantum measurement postulate, although a qubit may reside in a superposition of two orthogonal states as in (1.1), when we want to observe or measure its value, it collapses into one of the two orthogonal states $|0\rangle$ – with probability $|\alpha|^2$ – and $|1\rangle$ – with the probability $|\beta|^2$. After its measurement/observation, the original quantum state collapses to the measured state. Hence, the measurement irreversibly alters the original qubit state [36]. It is also worth noting that the measurement of a qubit state may also be carried out in a basis different from that in which the qubit was prepared in [37, 28, 38]. In the above description, for the sake of clarity, we assumed the standard basis also for the measurement.

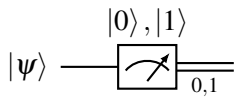


Figure 1.5: Quantum circuit for measuring a qubit. Single wires denote quantum states, whereas double wires denote classical states, namely, bits. The measurement of a qubit – whose output is a classical bit – induces the state of the qubit to collapse into the measured state.

Algorithms

There exists several quantum algorithms known or expected to outperform classical algorithms for problems spanning different areas, including cryptography, search and optimization, simulation of quantum systems and learning [12]. Remarkably, most known quantum algorithms use a combination of algorithmic paradigms – namely, sub-routines – specific to quantum computing [33]. These paradigms include the

Quantum Fourier Transform (QFT) [2], the Grover Operator (GO) [39], the Harrow/Hassidim/Lloyd (HHL) method for linear systems [40], Variational Quantum Algorithms (VQA) [41], and direct Hamiltonian simulation (SIM). A prominent example is Shor's algorithm for integer factorization [2], which is based on QFT, illustrated by the quantum circuit in Figure 1.6.

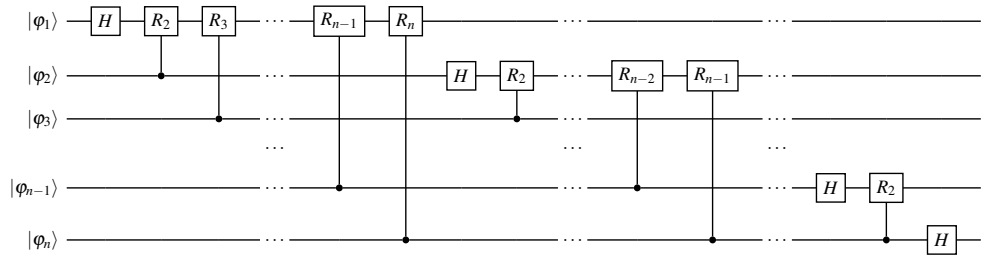


Figure 1.6: Quantum Fourier Transform (QFT) circuit. The i-th qubit is obtained through an Hadamard gate followed by $n-i$ controlled $\mathtt{R}_n$ operations – with $\mathtt{R}_i = \mathtt{P}_{2\pi/i}$ denoting the phase gate given in Table 1.1 – with the controlled operations controlled by the $n-i$ higher-order qubits.

For most practical applications, quantum algorithms require large quantum computing resources – in terms of qubit number – much larger than those available with current noisy intermediate-scale quantum (NISQ) processors. For example, the recently announced IBM Quantum Osprey device has 433 qubits, which is an impressive progress with respect to state-of-the-art quantum processors, but not yet sufficient, as an example, for running practical implementations of Shor's algorithm. Factoring $L = 2048$ bit primes – for breaking current RSA implementations – would require about $3L = 6144$ noise-free qubits [28]. It is worth noting that merely increasing the number of physical qubits is not sufficient, as some sort of quantum error correction [42] is also required to guarantee high-quality – namely, noise-free – computations.

Physical qubits

As already mentioned in Section 1.1.1, a variety of technologies have been investigated to implement physical qubits. The first aspect that differentiates these technologies is the way they encode the $|0\rangle$ or $|1\rangle$ state. Then, of course, there are the measurement processes and the available gates.

As an example, the polarization vector of a photon may lie anywhere in a 2D plane perpendicular to the motion direction of the photon. Therefore, its polarization state can be seen as the superposition of two basis states.

Electrons are subatomic particles characterized by a negative charge. Their spin orientation is defined by a clockwise or counter clockwise rotation, respectively spin-up or spin-down. Using magnetic fields, one can make electrons switch between the two spin orientation. Atoms too have their spin orientation and qubits can even be defined as their outer electron orbital position.

Atoms, electrons and such do not travel but interact very easily. For this reason, they are suitable for storing and processing quantum information. Photons move very easily and are useful for quantum information transmission.

The following list provides an overview of the main technologies explored so far:

- trapped ion qubits [43];
- linear optical qubits [44];
- neutral atoms trapped in an optical lattice [45];
- electrically defined or self-assembled quantum dots [46, 47];
- quantum dot charge based semiconductor qubits [48];
- rare-earth-metal-ion-doped inorganic crystal based qubits [49, 50];
- molecular magnets [51];
- superconducting qubits based on Josephson junctions [52, 53].

Regardless of their implementation, physical qubits are non-isolated quantum systems subject to different sources of noise, *i.e.* they follow a non-unitary evolution denoted as *quantum decoherence*. One can think about decohering evolution as a combination of unitary evolutions, each one influencing the qubit with an assigned probability. Decoherence is an irreversible process that brings a pure state into a mixed one.

Although universal quantum computers with thousands qubits are still far at the horizon, we are now entering in the era of *Noisy Intermediate-Scale Quantum* devices [8]. These *NISQ* devices have limited coherence time, a reduced number of qubits and imperfect gate implementation. Moreover, connections between qubits in this devices are limited, meaning that two-qubit gates can be placed only between connected qubit. This is certainly something to keep in mind when performing quantum computation on real hardware. Merely increasing the number of qubits in a single NISQ device is not sufficient to enable large-width quantum circuits, because more qubits means also more noise. Future devices will use *quantum error correction* (QEC) to obtain a few noiseless logical qubits from many noisy physical qubits. However, despite some recent relevant progresses [54], the road toward effective and efficient QEC is still long. With current NISQ devices, an alternative approach for executing large-width quantum circuits is possible. That is, distributed quantum computing.

1.1.2 Distributed Quantum Computing

NISQ devices are characterized by few hundreds of quantum bits (qubits) with non-uniform quality and highly constrained physical connectivity. Hence, the growing demand for large-scale quantum computers is motivating research on distributed quantum computing architectures [13, 14] as a scalable approach for increasing the number of qubits available for computational tasks, and experimental efforts have demonstrated some of the building blocks for such a design [55]. Indeed, with the network and communications functionalities provided by the *Quantum Internet* [15, 16, 17, 18, 19, 20, 13, 14, 21, 22], 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 [56] – for executing computational tasks that each NISQ device cannot handle by itself.

However, moving from local to distributed quantum computing implies crucial and specific challenges [55, 56]. A first issue that arises with distributed quantum computing is whether a given algorithm – equivalently, a given quantum circuit – is natively suitable for distributed execution. More specifically, a *perfectly distributable* quantum algorithm is a quantum algorithm that can be split into autonomous parts that do not interact – or, at least, weakly interact – with each others. If this is the case, each part can be assigned to some quantum processor, and each processor can contribute autonomously to the overall computation without introducing communication overhead for interacting with other processors.

Unfortunately, this is rarely the case. Let us consider the QFT algorithm, whose circuit is shown in Figure 1.6. The QFT is a core sub-routine in many quantum algorithms – e.g., Shor's algorithm and the quantum phase estimation algorithm – as mentioned above. From Figure 1.6, it is clear that QFT requires each qubit to strongly interact with all the other qubits through controlled $\mathtt{R}_n$ gates. Hence, QFT can be considered as the archetype of monolithic quantum algorithms, namely, of an algorithm not natively-suitable for distributed execution.

In general, when moving from local to distributed quantum computing one faces two main challenges, namely, quantum algorithm partitioning and execution management. To partition a monolithic quantum algorithm, a quantum compiler must be used to find the best breakdown, i.e., the one that minimizes the number of gates that are applied to qubits stored at different devices. The problem of Quantum Compilation is treated in more detail in Sections 1.2.1 and 1.2.2. Here we briefly review some literature that addresses the partitioning of relevant quantum algorithms, using techniques that are tailored to the specific considered algorithms rather than general-purpose. These works may represent a good reference for a comparative evaluation of quantum compilers.

In [57], Neumann et al. present two distribution schemes for the *quantum phase estimation* algorithm, give the resource requirements for both and show that using less noisy shared entangled states results in a higher overall fidelity. Introduced by

Kitaev [58], the quantum phase estimation algorithm returns an approximation of an eigenvalue of a given unitary U and a corresponding eigenvector. It has numerous applications, including Shor's algorithm [2]. The solution proposed by Neumann et al. is based on the distributed version of the QFT circuit, obtained by means of non-local controlled U-gates. A non-local controlled U-gate is a generalized controlled gate between qubits residing in different QPUs [59], also known as a `Telegate`. `Telegate` operations and their implementation are discussed in Section 1.1.3.

Another example of distributable quantum algorithm is the *Variational Quantum Eigensolver* (VQE), a VQA that can be used to estimate ground state energies of molecular chemical Hamiltonians. In [60], Di-Adamo et al. provide a *Local to Distributed Circuit* algorithm that, given a circuit representation as a series of layers and a mapping of qubits, searches for any control gates where the control and target are physically separated between two QPUs. When found, the algorithm inserts, between the current layer and next layer in the circuit, the necessary steps to perform the control gate in a nonlocal way. The size (maximum number of qubits) of the achievable Ansatz state for the VQE algorithm grows linearly with the number of QPUs, with slope linearly increasing with the number of qubits per QPU. The depth of the resulting quantum circuit is $\Omega(n)$, meaning it has a tight upper and lower bound proportional to the number n of qubits.

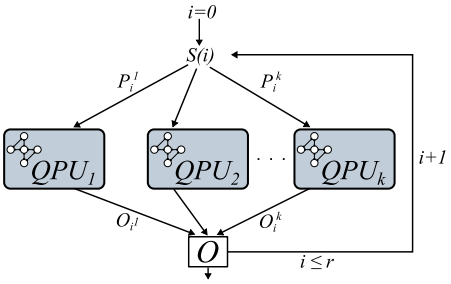


Figure 1.7: Execution of multiple quantum circuit instances with k QPUs. For each execution round i, a schedule $S(i)$ maps some quantum circuit instances to the quantum network – each QPU receiving a quantum circuit P_i^j that is either a monolithic one or a sub-circuit of a monolithic one. The classical outputs are accumulated into an output vector O.

In [61], the authors present a distributed adder and a distributed distance-based classification algorithm. Both applications are framed in a way where a quantum server and K other quantum nodes interact, with specific behaviors. In particular,

the server is responsible for orchestrating the computation by means of non-local CNOT gates, while the K parties provide inputs. It is possible to reframe these applications, such that the proposed quantum circuits are considered as monolithic and subsequently split in $K+1$ parts to be submitted for execution to a quantum network.

All of the above techniques are designed for the optimal partitioning of very specific algorithms and are not suitable for the distribution of any given quantum algorithm, thus leaving the need of a general-purpose compilation framework for DQC architechtures.

The other challenge that one must face in DQC is related to the execution management of distributed quantum computations. In general, given a collection $\mathscr{P}$ of quantum circuit instances to be executed, this collection should be partitioned into non-overlapping subsets $\mathscr{P}_i$, such that $\mathscr{P} = \cup_i \mathscr{P}_i$. One after the other, each subset will be assigned to the available QPUs. In other words, for each execution round i, there exists a schedule $S(i)$ that maps some quantum circuit instances to the quantum network. If DQC is supported, some quantum circuit instances may be split into sub-circuit instances, each one to be assigned to a different QPU, as illustrated in Figure 1.7). A QPU scheduling algorithm that partially address this service was proposed by Parekh et al. [62]. Such an algorithm is based on a greedy approach, trying to fill all available QPUs while minimizing the number of distributed quantum circuit instances. Here the partitioning of quantum circuit instances is arbitrary, not taking into account the features of the programs.

1.1.3 Quantum Networking

As hinted in the previous section, when it comes to distributed quantum computing, qubits are distributed among multiple smaller quantum processors, interconnected by some sort of quantum network.

Accordingly, whenever a quantum gate must operate on *remote qubits* – namely, qubits located in different quantum processors – some sort of *communication primitive* must be available for performing remote operations. Unfortunately, this communication primitive cannot be easily accomplished through classical protocols. Indeed, the different physical phenomena underlying quantum communications impose

a paradigm shift.

Thankfully, *entanglement* provides an invaluable tool for implementing remote operations without violating quantum mechanics [56]. As previously detailed in Section 1.1.1, 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 [63].

This peculiar property can be exploited to achieve the so-called quantum teleportation, which allows to transmit an unknown quantum state from one qubit to another without an actual physical transfer of the qubit storing the state. More specifically, quantum teleportation requires:

- an EPR pair, namely a pair of maximally entangled qubits such as the Bell state in Eq.(1.4), with one qubit of the pair located at the source QPU and the other qubit located at the destination;
- local quantum operations both at the source and at the destination;
- the transmission of two classical bits from the source to the destination.

The circuit representation of the quantum teleportation process is illustrated in Figure 1.8. Specifically, the source QPU performs a *Bell State Measurement* (BSM) on the qubit storing the quantum state to be transmitted and on the first of the entangled qubits. As represented in the gray box in the figure, The BSM, enclosed in the gray box in Fig. 1.8, consists of a `CNOT` gate – with the qubit to be teleported acting as control and the entangled qubit acting as target – followed by an Hadamard gate on the qubit to be teleported and, finally, a measurement of both the qubits. After the BSM, the source uses classical communication to transmit two classical bits, encoding the measurement outcomes of the BSM. It should be specified that, after the BSM, the source qubit has now lost the encoded state, which now resides at the destination, albeit some needed corrections. In fact, the teleported state may have been undergone a phase and/or a bit-flip. The measurement outcomes of the two qubits,

transmitted by the source QPU, can be used to determine whether these flip events occurred. Thus, the destination performs a post-processing to reconstruct the original state $|\psi\rangle$, as detailed in Table 1.9.

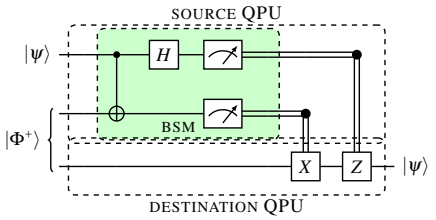


Figure 1.8: The quantum teleportation circuit. The first two wires belong to the source QPU, whereas the bottom wire belongs to the destination QPU. Each double line denotes the transmission of one classical bit – i.e., the measurement outcome – between the remote processors. The two classical bits are used as detailed in Table 1.9 to determine the execution of the two conditional gates `X` and `Z`. In the end, the original state $|\psi\rangle$ is located at entangled qubits available at the destination.

Measurement Outcomes	Correction operations
00	`I`
01	`X`
10	`Z`
11	`ZX`

Figure 1.9: Quantum teleportation correction operations performed at the destination to recover the original quantum state. The less significant bit representing the outcome of the entangled qubit measure in Figure 1.8.

One may observe that direct transmission of qubits is still needed to distribute entangled states among the network nodes. However and as deeply clarified in [63], differently from unknown qubits, entangled states can be repeatedly prepared for facing with losses and/or noise corruptions.

Quantum teleportation stands at the foundations of the communication primitives known as `TeleData` and `TeleGate` [64], in the distributed quantum computing context.

Before diving into the details of `TeleData` and `TeleGate`, we must introduce a distinction between *communication qubits* and *data qubits* [56, 65]. Specifically, for each QPU a subset of qubits is reserved for inter-processor communication, we refer to those as *communication qubits* [66], as opposed to the remaining qubits dedicated

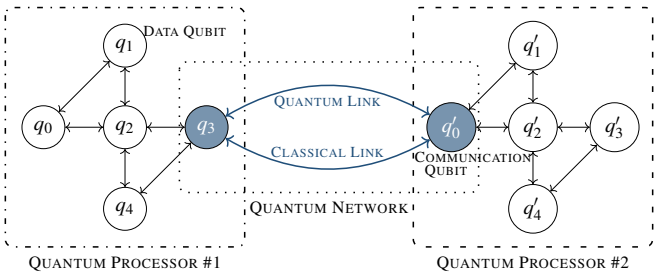
to processing or storage, namely *data qubits*.

To distribute entangled states between different quantum processors, at least one communication qubit at each processor must be reserved for remote inter-processor operations. Hence, a crucial trade-off between communication and data qubits arises. Specifically, for each `TeleData` and `TeleGate`, an entangled state is consumed and a new one must be distributed between the remote processors through the quantum link before another inter-processor operation can be executed. Hence, the more communication qubits are available within a processor, the higher the entanglement rate and the more inter-processor operations can be executed in parallel, reducing the overhead induced by the distributed computation. But the more communication qubits are available for inter-processor communication, the less valuable resources – i.e., data qubits – are available for computing.

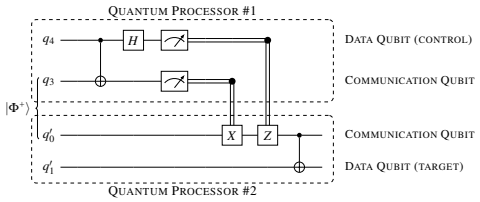
As an example, consider two quantum processors interconnected via a quantum network as depicted in Figure 1.10. Qubits q_3 and $q_0^{'}$ are communication qubits and any interaction between the two remote processors is carried out by exploiting them via either a `TeleData` or a `TeleGate` process.

With a `TeleData`, quantum information stored within a data qubit at the first processor, say $|\varphi\rangle$ in q_4 in Figure 1.10a, is teleported into a communication qubit of the second processor, say $q_0^{'}$ in the same figure. Once the quantum state $|\varphi\rangle$ is teleported in $q_0^{'}$, any remote operation – originally involving q_4 and some data qubits at the second processor – can be now implemented through local operations as shown with the last `CNOT` in Figure 1.10b. It must be noted, though, that whether the teleported quantum state should subsequently interact with data qubits at the first processor, a new teleportation process must be performed for teleporting the quantum state back to the first processor. `TeleData` is not the only available option for implementing remote operations. In fact, a `TeleGate` enables to execute a direct gate between qubits belonging to remote processors by exploiting again entanglement. For instance, a remote `CNOT` with data qubit q_4 and q_0^1 in Figure 1.10a acting as control and target, respectively, can be implemented with local `CNOT`s at each quantum processor, as shown with the quantum circuit in Figure 1.10c.

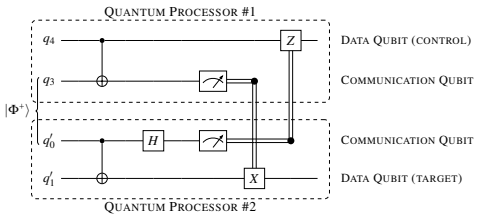
From Figure 1.10, one might assume that distributed quantum computing requires



(a) Two quantum processors interconnected through a quantum network, composed by a classical and a quantum link. The classical link is used to transmit classical information, whereas the quantum link is needed for distributing entangled states between the two remote processors. At least one physical qubit at each processor must be reserved for entanglement generation. This kind of qubits – dark-blue-colored in the figure – are the *communication qubits* to distinguish them from the *data qubits* – white-colored in the figure.



(b) `TeleData`. To perform a `TeleData` between remote processors – say to move the quantum state $|\varphi\rangle$ stored by data qubit q_4 in Figure 1.10a to communication qubit $q_0^{'}$ – a Bell state such as $|\Phi^+\rangle$ must be distributed through the quantum link so that each pair member is stored within the *communication qubit* at each processor. Once $|\varphi\rangle$ is teleported at $q_0^{'}$ (with local quantum operations and classical transmission), the remote operation – for instance, a `CNOT` with $|\varphi\rangle$ as control and the state stored by qubit $q_1^{'}$ as target as shown with the last `CNOT` in the figure – can be executed through local operations.



(c) `TeleGate`. A `TeleGate` enables a direct gate between remote physical qubits stored at different processors without the need of quantum state teleportation, as long as a Bell state such as $|\Phi^+\rangle$ is distributed through the quantum link. For instance, a remote `CNOT` between q_4 and $q_1^{'}$ in Figure 1.10a can be implemented with two local `CNOT`s 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.

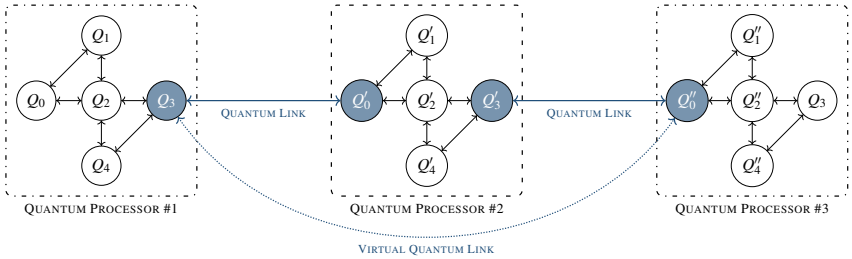
Figure 1.10: Remote operations through either `TeleData` or `TeleGate`. Figure 1.10a shows the network topology along with the processors coupling maps, whereas Figures 1.10b and 1.10c illustrate the quantum circuit detailing the classical (2 bits) and the quantum (the Bell state) resources needed to execute a `TeleData` and a `TeleGate`, respectively.

a fully-connected network topology, namely, that each quantum processor must be directly inter-connected with all the other processors. In other words, it might seem that the connectivity between quantum processors strongly dependent on the availability of a direct entanglement generation and distribution architecture. On the contrary, distributed quantum computing can exploit a strategy, called *entanglement swapping* [55] and illustrated in Fig 1.11, to create entanglement between qubits stored at remote processors, even if the processors are not directly connected through a quantum link.

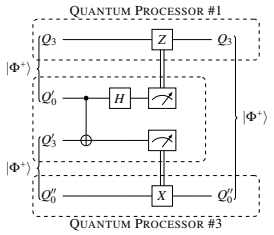
In a nutshell, to distribute a Bell state between remote processors – say quantum processor #1 and #3 in Fig. 1.11a – two Bell states must be first distributed through the quantum links so that one Bell state is shared between the first processor and an intermediate node and another Bell state is shared by the same intermediate node and the second processor. Then, by performing a Bell state measurement (consisting of a `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. 1.11b – a Bell state is obtained at the remote communication qubits – i.e., qubits Q''_0 and Q_3 in Fig. 1.11a – 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. For instance, qubit Q_4 in Fig. 1.11a can interact with just two qubits within the same processor via local `CNOT`s and two qubits within the neighbor processor via remote `CNOT`s. However, it can interact with two more qubits – i.e., Q''_1 and Q''_2 – via entanglement swapping. And the higher the number of available quantum processors, the higher 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.

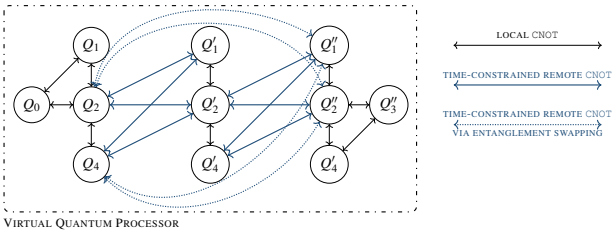
However, it must be acknowledged that the augmented connectivity provided by entanglement swapping does not come for free. Indeed, entanglement swapping con-



(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 Figure 1.11a. The solid blue lines denote direct connections between adjacent processors, whereas the dotted blue lines denote connections between distant processors achievable via entanglement swapping.

Figure 1.11: Augmented connectivity. Entanglement swapping increases the connectivity between physical qubits, with a number of possible connections that scales at least linearly with the number of processors.

---

sumes 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.

1.2 Quantum Compiling

The problem of *quantum compilation*, i.e., device-aware implementation of quantum algorithms, is a challenging one. A good quantum compiler must translate an input quantum algorithm into the most efficient equivalent of itself [9], getting the most out of the available hardware. With this definition in mind, one could see the compiler as an intermediary between the user and the hardware. As a matter of fact, while designing a quantum algorithm using the quantum circuit formalism, the designer is generally focused on the logic of the circuit expressing the computation required by the algorithm, regardless from the particulars of the quantum hardware that will execute the circuit. This *abstract* circuit is then mapped to a circuit to be executed on a specific quantum hardware by means of a suitable compiler.

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 [67, 68]. The rationale is to keep the execution time of the quantum circuit within the coherence time of the underlying quantum hardware architecture [28, 36]. 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 exists 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 [55]. 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*.

A universal quantum gate set [69] is any set of gates that any operation possible on a quantum computer can be reduced to. For example, IBM quantum processors are realized exploiting the superconducting technology, and any logical gate that can be run on current IBM quantum processors is built from a gate set composed by the

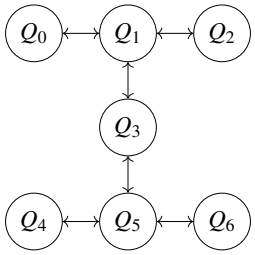


Figure 1.12: Coupling map of the *ibm_perth* quantum processor [71]. The seven physical qubits are represented by circles. The arrows denote the possibility to realize a two-qubit `CNOT` gate between the connected qubits, with the arrow pointing toward the target qubit. As an example, a `CNOT` between 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.

controlled-not (`CNOT`) gate and two single-qubit gates [70]. Furthermore, regardless of the underlying qubit technology, any quantum processor exhibits physical constraints 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 Figure 1.12 with the *coupling map* of an IBM quantum processor. The rationale for these limitations is as follows. Multi-qubit gates require the qubits to interact with each other, for instance through directly coupling with microwave resonators as in superconducting qubits [72, 73, 74]. However, the higher is the number of interconnections, the harder is to preserve the quantum information encoded within a physical qubit from noise effects. Hence, limitations on the number of interconnections that can be realized within a quantum processor – e.g., limitations on the pair of physical qubits that can undergo a `CNOT` gate – arise as a consequence of the noise (and the physical-space) constraints.

Compilation strategies are usually composed by sequential *passes* that perform placement, routing and optimization tasks. Placement is performed once, at the very beginning of the compiling process. It is the task of defining a mapping between the virtual qubits of the input quantum circuit and the physical qubits of the device. Routing is the task of modifying the circuit in order to move through subsequent mappings, by means of a clever swapping strategy, in order to conform to the qubit

layout of the device. Optimization is the task of minimizing some property of the circuit in order to reduce the impact of noise. Several routing and optimization passes may be executed within a compilation strategy.

Noise-adaptive compilers do take the noise statistics of the device into account [75, 76, 77, 78], for some or all passes. The noise statistics can be obtained from calibration data, and updated after each device calibration. For example, Qiskit [79] allows for programmatic retrieval of IBM Q devices' calibration data.

As an example of compliance with the hardware architecture, consider the problem of compiling the circuit in Fig. 1.13 onto a specific device. One could choose a trivial initial mapping of virtual qubits to the physical ones, such as the one depicted in red to the left circuit in Fig. 1.13. However, with such a mapping, the CNOT between q_0 and q_3 could not be directly executed on the device. A more suitable mapping is instead the one shown in blue to the right circuit in Fig. 1.13, as it enables the execution of all CNOTs onto the device at the cost of inserting only one SWAP gate.

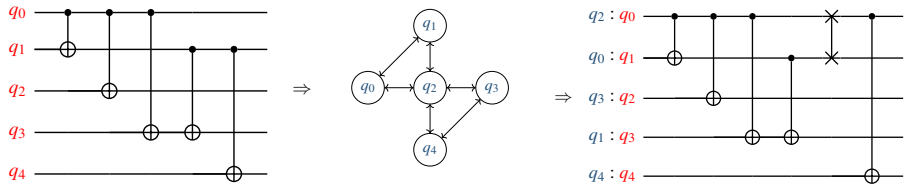


Figure 1.13: Example of quantum compiling. The circuit on the left is translated into the circuit on the right, in order to cope with the coupling map in the center. Within the rightest figure, the q_i with purple font denotes the physical qubits assigned to the logical qubits q_j with black font. The SWAP gate– represented by two $\times$ symbols interconnected by a vertical line – introduced between logical qubits q_1 and q_2 swaps their quantum states, so that the last `CNOT` gate can be applied between two neighbor physical qubits.

In the literature, compiled circuits are frequently evaluated in terms of depth and gate count overhead with respect to the input circuits [80]. Calculating these figures of merit does not require to execute the compiled circuit. Another common method is to run the compiled circuit many times with a figure of merit being the success rate, i.e., the fraction of runs that resulted in a correct (classical) answer. The Hellinger fidelity

is a measure of the distance between two random distributions. It is frequently used to compare the sampled distribution of the results of a quantum computation to the theoretical distribution (provided that the latter one is known).

Recently, Mills et al. [81] have presented a framework for application-motivated benchmarking of full quantum computing stacks. The benchmarks defined there have a *circuit class*, describing the type of circuit to be run on the system, and a *figure of merit*, quantifying how well the system did when running circuits from that class. The idea is that a circuit class should represent a particular application domain. The application-motivated circuit classes proposed by Mills et al. draw inspiration from quantum algorithmic primitives [82] and from the literature on near-term quantum computing applications (e.g., machine learning and chemistry).

Intuitively, a circuit transformation may introduce some overhead, in terms of number of operations and noise. In DQC architectures, there is also a non-negligible communication cost, as discussed in Section 1.1.3. Therefore, the compiler faces an optimization problem, i.e., finding a feasible transformation while minimizing the overhead. In general, the quantum compilation problem is NP-Hard [10, 11].

For DQC to be effective and efficient, the quantum compiler must find the best partitioning for the abstract circuit, i.e., the partitioning that minimizes the overall communication cost required to execute the distributed circuit (such as the one illustrated in Figure 1.14), then perform smart remote operation scheduling to optimize ebits use. At the same time, the quantum compiler should also find the best local transformation for each partition.

From the above, it should be clear that the design of an efficient compiler is not trivial. Most quantum compilers for DQC are characterized by two fundamental steps, namely *qubit assignment* or *circuit partitioning* – to distribute the qubits of the abstract circuit between the different QPUs – and *non-local gate handling* – to minimize the communication cost via circuit transformations and gate scheduling.

1.2.1 Compilation for Local Quantum Computing

Recently, some noteworthy quantum compiling techniques for local quantum computing have been proposed. Here we survey those that have been implemented into

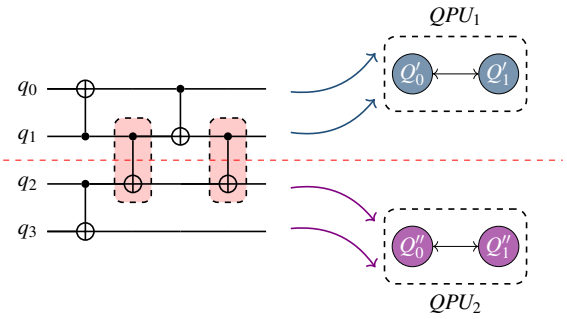


Figure 1.14: Example of circuit partitioning. The circuit is partitioned in two. The first partition, containing qubits q_0 and q_1, is assigned to QPU_1 and the second partition, containing qubits q_2 and q_3, is assigned to QPU_2. As a consequence of the partitioning, some two-qubits gates – highlighted in red – must now be executed across different QPUs.

actual compilers, and benchmarked.

The approach proposed by Zulehner *et al.* [83] is to partition the circuit into layers, each layer including gates that can be executed in parallel. For each layer, a compliant CNOT mapping must be found, starting from an initial mapping obtained from the previous layer. Denoting the number of physical qubits as m and the number of logical qubits as n, in the worst case there are $m!(m-n)!$ possible mappings. Such a huge search space cannot be explored exhaustively. The A^* search algorithm is adopted, to find the less expensive swap sequence. Moreover, a lookahead strategy is adopted to minimize additional operations to switch between subsequent mappings. The proposed solution is efficient in terms of running time and output depth, but may not be scalable because of the exponential space complexity of the A^* search algorithm [84].

SABRE by Li *et al.* [85] is a SWAP-based bidirectional heuristic search method. It requires a preprocessing phase consisting of the following steps. First of all, the distance matrix over the coupling map is computed. Then, the directed acyclic graph that represents the two-qubit gate dependencies of the circuit is generated. A data structure denoted as F (front layer) is initialized as the set of two-qubit gates without unexecuted predecessors. The preprocessing phase ends up with the generation of a

random initial mapping. Then, the compiling phase consists in iterating the following steps over F, until F is empty. First, all executable gates are removed from F and their successors are added to F. Second, for those gates in F that cannot be executed, the best SWAP sequence is selected using an heuristic cost function based on distance matrix. Experiment results show that SABRE can generate hardware-compliant circuits with less or comparable overhead, with respect to the approach proposed by Zulehner *et al.* [83].

In Qiskit (version 0.20) [79], the compiling process is implemented by a customizable Pass Manager that schedules a number of different passes: layout selection, unrolling (i.e., gate synthesis), swap, gate optimization, and more. Four swap strategies are currently available: Basic, Stochastic, Lookahead and SABRE. The Stochastic strategy uses a randomized algorithm to map the input circuit to the selected coupling map. This means that a single run does not guarantee to produce the best result. Qiskit's NoiseAdaptiveLayout placement pass associates a physical qubit to each virtual qubit of the circuit using calibration data, based on the method proposed by Murali et al. [75]. The pass maps virtual qubit pairs in order of decreasing frequency of the CNOT occurrences between them. If a pair exists with both qubits unmapped, the pass picks the best available physical qubit pair, based on CNOT reliability, to map it. If a pair has only one qubit unmapped, the pass maps that qubit to a location that ensures maximum reliability for CNOTs with previously mapped qubits. In the end if there are unmapped qubits, the pass maps them to any available physical qubit.

Nishio et al. [77] proposed a placement pass and a routing pass. The placement pass, which is denoted as Greatest Connecting Edge Mapping, leverages a strategy that is very similar to the one proposed by Murali et al. [75]. The routing pass is a beam search algorithm with an heuristic cost function based on the estimated success probability of candidate SWAP gates.

Recently, Niu et al. [76] have implemented a hardware-aware routing pass, inspired by the work of Li et al. [85], that iteratively selects the best scoring SWAP with respect to calibration data as well as qubits distance. The influence of each of these factors on the cost function can be set by means of tunable weights.

One of the most advanced quantum compiler is t|ket⟩ [78], which is written in C++. The compiling process proceeds in two phases: an architecture-independent optimization phase, which aims to reduce the size and complexity of the circuit; and an architecture-dependent phase, which prepares the circuit for execution on the target machine. The architecture-independent optimization phase consists of peephole optimizations (targeting small circuit patterns) and macroscopic optimizations (aiming to identify high-level macroscopic structures in the circuit). The architecture-dependent one searches for candidate partial placements of virtual qubits to the physical ones. This is done by casting the problem as finding a subgraph monomorphism between the connectivity graph and a graph representing virtual qubit CNOT interactions in the circuit. Where different possible candidates placement are found, the pass chooses the one with the maximum expected overall fidelity. The end product of this process is a circuit that can be scheduled for execution by the runtime environment, or simply saved for later.

1.2.2 Compilation for Distributed Quantum Computing

As previously mentioned, quantum compilation for DQC is characterized by two fundamental steps, *qubit assignment* (or *circuit partitioning*) and *non-local gate scheduling*. In DQC, qubit assignment is generally tackled as a partitioning problem. Specifically, for a given set of virtual qubits, one needs to choose a partition that maps sub-sets of logical qubits to processors, while minimizing the number of required interactions among different sub-sets, as depicted in Figure 1.14. The main goal is to minimize the number of consumed *ebits* – i.e., EPR pairs shared between QPUs –, as it is the main bottleneck to distributed quantum computation. To this aim, qubit assignment discussed above represents a starting point for further optimization steps, which now concern circuit manipulation.

Andrés-Martínez and Heunen [86] use cat-entanglement to implement non-local quantum gates. *Cat-entanglement* is a process introduced by Yimsiriwattana et al. [87], which is substantially equivalent to the telegate introduced in Section 1.1.3. The chosen gate set contains every one-qubit gate and a single two-qubit gate, namely the CZ gate (i.e., the controlled version of the Z gate). The authors consider no restriction on

the ebit connectivity between QPUs. Then, they reduce the problem of distributing a circuit across multiple QPUs to hypergraph partitioning. The proposed approach is evaluated against five quantum circuits, including QFT. The proposed solution has some drawbacks, in particular that there is no way to customize the number of communication qubits of each QPU.

Sundaram et al. [88] present a two-step solution, where the first step is qubit assignment. Circuits are represented as edge-weighted graphs with qubits as vertices. The edge weights correspond to an estimation for the number of *cat-entanglements*. The problem is then solved as a minimum k-cut, where partitions have roughly the same size. The second step is finding the smallest set of cat-entanglement operations that will enable the execution of all telegates. The authors state that, in a special setting, this problem can be reduced to a vertex-cover problem, allowing for a polynomial-time optimal solution based on integer linear programming. They also provide a $O(\log n)$-approximate solution, where n is the total number of global gates, for a generalized setting by means of greedy search algorithm. In [89], the same authors extend their approach to the case of an arbitrary-topology network of heterogeneous quantum computers by means of a Tabu search algorithm.

In [90], by Daei et al., the circuit becomes an undirected graph with qubits as vertices, while edge weights correspond to the number of two-qubit gates between them. Then, the graph is partitioned using the Kernighan-Lin (K-L) algorithm for VLSI design [91], so that the number of edges between partitions is minimized. Finally, each graph partition is converted to a quantum circuit.

In [92], the authors represent circuits as bipartite graphs with two sets of vertices – one set for the qubits and one for the gates – and edges to encode dependencies of qubits and gates. Then, for the qubit assignment problem, they propose a partitioning algorithm via dynamic programming to minimize the number of `TeleData` operations.

Dadkhah et al. [93] propose a heuristic approach to replace the equivalent circuits in the initial quantum circuit. Then, they use a genetic algorithm to partition the placement of qubits so that the number of teleportations could be optimized for the communications of a DQC.

Nikahd et al. [94] exploit a minimum k-cut partitioning algorithm formulated as an ILP optimization problem, to minimize the number of remote interactions. They use a moving window and apply the partitioning algorithm to small sections of the circuit, thus the partition may change with the moving window by means of teledata operations.

Cuomo et al. in [95] model the compilation problem with an Integer Linear Programming formulation. The formulation is inspired to the vast theory on dynamic network problems. Authors managed to define the problem as a special case of *quickest multi-commodity flow*. Such a result allows to perform optimization by means of techniques coming from the literature, such as a *time-expanded* representation of the distributed architecture.

Chapter 2

General-Purpose Quantum Compilation Framework

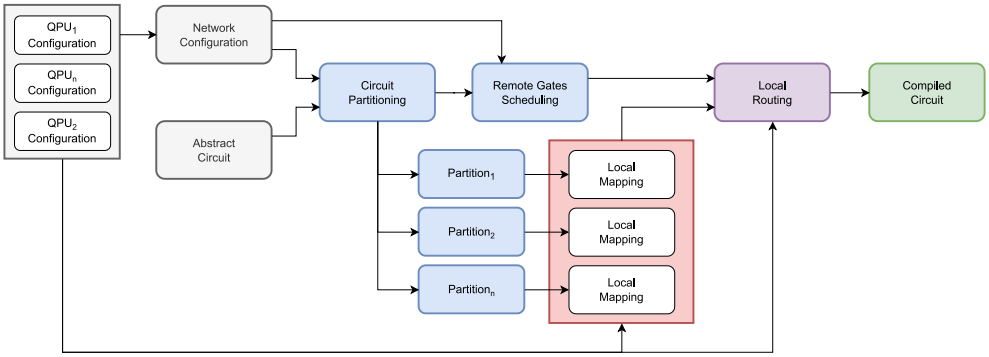


Figure 2.1: Workflow of the proposed general-purpose quantum compilation framework for DQC architectures.

As mentioned in Section 1.1.2, there is a lack of a general-purpose framework for compiling quantum circuits to DQC architectures. Such a framework should be circuit agnostic, i.e., able to compile any circuit to any suitable DQC architecture. Moreover, this framework should bridge the gap between local compilation and compilation for DQC. Current proposals from the literature tackle the problems of cir-

cuit partitioning and remote gate scheduling but do not take into account the local connectivity of each QPU. Our proposal for a general-purpose quantum compilation framework is shown in Fig. 2.1.

The Chapter is organized as follows. We first detail the input required by the proposed framework and the produced output. Then we illustrate some techniques for local mapping and circuit partitioning, both from previous work and the literature, that could be exploited by the proposed framework. We continue by providing an overview of different techniques for local routing and remote gates scheduling. Finally, we introduce a strategy for bridging the gap between local compilation and compilation for DQC.

2.1 Input & Output

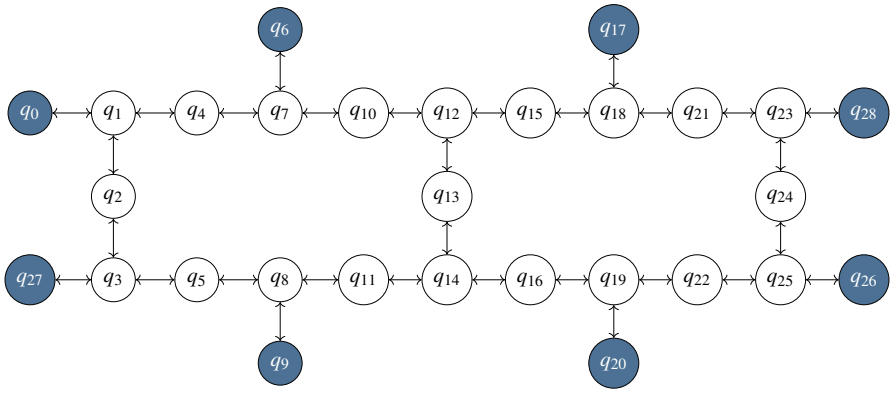


Figure 2.2: QPU configuration with 20 *data qubits* and 8 *communication qubits*, inspired by IBM's heavy hexagon devices [71].

The proposed quantum compilation framework takes as input a quantum circuit and a network configuration. The network configuration describes how QPUs are connected in the targeted DQC architecture, including quantum channels capacity, i.e., the number of communication qubits for each channel. The network configuration should includes descriptions of the internal configurations of the QPUs, i.e. the

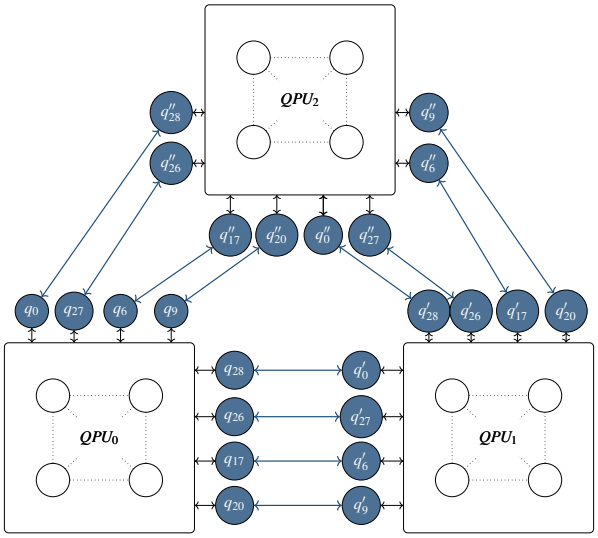


Figure 2.3: DQC architecture comprising 3 QPUs as shown in Fig. 2.2. Each QPU is connected to the others and each QPU supports up to 4 *communication qubits* per connection.

coupling map and the set of available data qubits and communication qubits.

The output of the framework is of course a compiled circuit. This circuit can either be a unique object containing all properly scheduled local and remote gates or a collection of sub-circuits, one for each QPU. In the first case, each QPU should be able to filter gates that pertains to them.

2.2 Mapping

A typical compilation strategy is composed of sequential *passes* that perform placement, routing and optimization tasks. Mapping is the task of mapping the virtual qubits of an input circuit, to the physical qubits of the device that will carry out the computation. Mapping should facilitate the subsequent routing pass. From the DQC perspective, this problem translates into partitioning a circuit and assign each partition to a different QPU, with the aim of minimizing the number of interactions, i.e. gates, across different partitions, i.e., QPUs.

2.2.1 Local Mapping

In a previous work [96], we tackled the problem of mapping to IBM Q devices [97] the RyRz circuits used to compute, with the variational quantum eigensolver (VQE) method, the ground state properties of molecular systems. These circuits were introduced for the first time in [98] as a heuristic hardware-efficient wavefunction Ansätz for the calculation of the electronic structure properties of small molecular systems such as hydrogen H_2, lithium hydride, LiH, and berillium hydride, BH_2, on a quantum computer. Contrary to other quantum circuits inspired by classical wavefunction expansion techniques (e.g., the coupled cluster expansion [99, 100]), in this case the nature of the circuit is solely motivated by the requirement of producing an entangled wavefunction for the many-electron systems that optimally fits the connectivity of the hardware at disposal. In most cases, the RyRz circuits offer a well balanced compromise between these two requirements. These circuits, when implemented with full entanglement (Fig. 2.4), are characterized by repeated sequences of a pattern that we denote as *inverted CNOT cascade*.

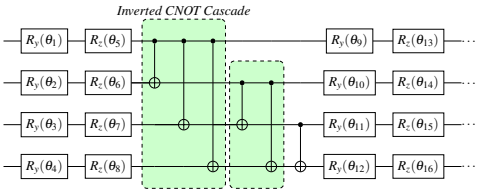


Figure 2.4: RyRz circuit example.

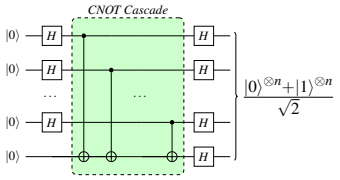


Figure 2.5: GHZ circuit.

The pattern characterizing RyRz circuits is very similar to the one highlighted in Fig. 2.5, which plays a prominent role in several quantum algorithms such as the one used to produce GHZ states [101, 102]. Indeed an *inverted CNOT cascade* can be turned into the other one by inverting all of its CNOT gates by means of H gates on both control and target qubits before and after the CNOT. Of course adding H gates to invert a CNOT would alter the circuit identity, therefore, instead of adding an H gate, two H gates are added so that they can negate each other's effects and leave the circuit identity untouched. The result of this operation is shown in Fig. 2.6a.

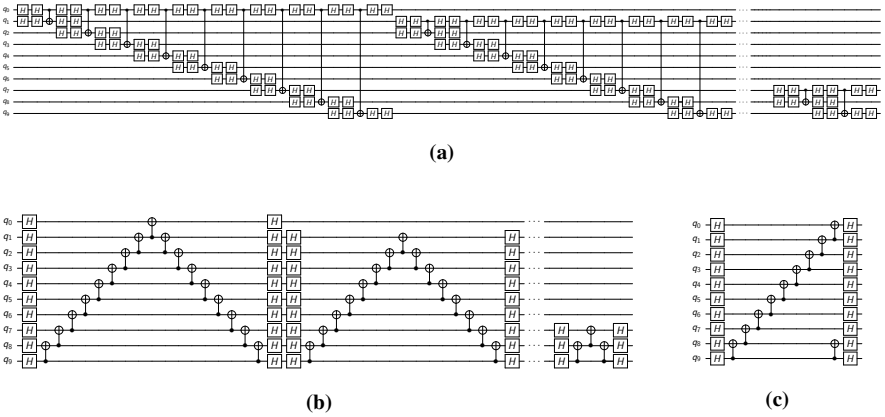
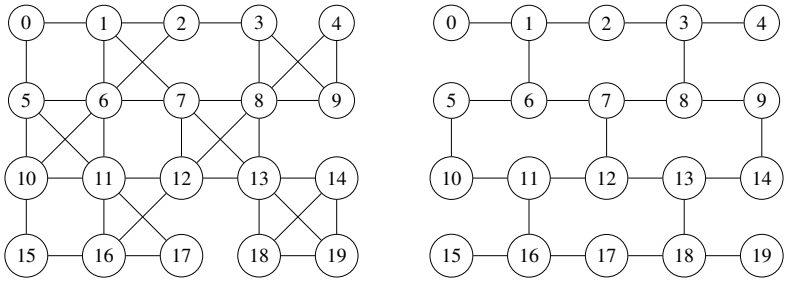


Figure 2.6: (a) Circuit with multiple inverse CNOT cascades after CNOT inversion. **(b)** Circuit with multiple inverse CNOT cascades after nearest-neighbor decomposition. **(c)** Circuit with multiple inverse CNOT cascades after gates cancellation.

The coupling maps in Fig. 2.7 prevent from placing all CNOT gates like in the ideal GHZ circuit, i.e., making a *CNOT cascade* where $n-1$ qubits control the nth qubit. It is indeed possible to turn the ideal GHZ circuit into an equivalent one characterized by a unique sequence of CNOT gates. It would only be a slight change to the technique discussed in a previous work [103].



(a) **(b)**

Figure 2.7: (a) 20 qubits *ibmq_tokyo* and **(b)** *ibmq_almaden* [97].

However, this technique works only if the aim is to produce a GHZ state starting

from a $|0\rangle^{\otimes n}$. A possible solution is to exploit the nearest neighbor decomposition for a uniformly controlled gate studied by Tucci [104]. The only requirement for Tucci's decomposition is that qubits are to be arranged in a linear chain, as shown in Fig. 2.8.

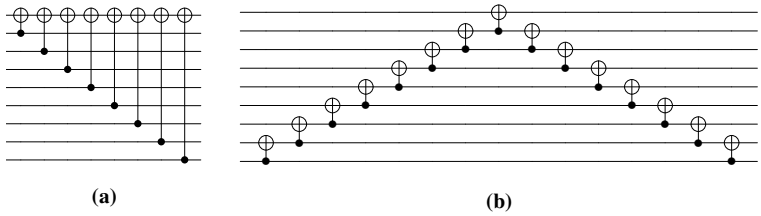


Figure 2.8: (a) CNOT cascade. **(b)** Decomposition of a CNOT cascade

The result of such decomposition on RyRz circuits is depicted in Fig. 2.8b. Finally, the circuit can be optimized, as a CNOT gate is the inverse gate of itself, as well as the H gate, producing the circuit shown in Fig. 2.6c.

Clearly the sequence of gates shown in Fig. 2.6c, where every qubit q_i controls q_{i+1}, cannot be directly executed on the coupling map in Fig. 2.7b, as q_i is not always connected with q_{i+1}.

A possible solution is to find a path in the coupling map such that every qubit q_i, with $1 < i < n-1$, with n being the number of qubits in the device, has a connection with its nearest neighbors q_{i-1} and q_{i+1}. This is related to the problem of finding an *Hamiltonian path*, i.e., a path that visits each vertex of a graph exactly once. The Hamiltonian path problem is a special case of the *Hamiltonian cycle* problem and is known to be NP-Complete, as it is an instance of the famous *traveling salesman* problem [105].

Fortunately, one can take advantage of the features of the coupling map such as its regular structure and the fact that every qubit is identified by a number ranging from 0 to n. As each undirected link can be seen as a couple of ingoing and outgoing links, the path obtained resembles a *chain* and will be denoted as such, from now on.

Algorithm 1 computes a chain in an undirected graph $\mathscr{G}$ starting from node 0, where $\mathscr{C}$ is the chain initialized as empty, $\mathscr{S}$ is the set of nodes to be explored and $\mathscr{N}_x$ is the set of neighbors of node x. The algorithm loops over the nodes until the set

of explored nodes $\mathscr{E}$ is equal to the set of nodes in $\mathscr{G}$. For every node added to $\mathscr{C}$, the chain() algorithm checks if the node's neighbors lead to a dead end, i.e., are isolated. If one neighbor is found to be isolated, it is added to the set of isolated nodes $\mathscr{I}$ and also to $\mathscr{E}$.

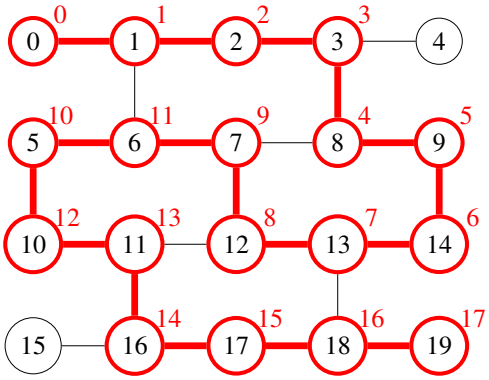


Figure 2.9: Qubit chain in *ibmq_almaden* highlighted in red.

After executing Algorithm 1 on the coupling map in Fig. 2.7b, the path obtained can be used to formulate an initial layout for the circuit in Fig. 2.6c, such that logical qubit q_i corresponds to chain element $\mathscr{C}[i]$. Fig. 2.9 shows the path obtained in the coupling map highlighted in red. Such an initial layout eliminates the need to use SWAP gates and produces a circuit with ideally no increase in depth.

Nodes 4 and 15 are left outside the chain as it is already sufficiently long, to map the circuit in Fig. 2.6c, and adding these two extra qubits would require the use of SWAP gates during compilation. For a larger circuit, then those nodes will be inserted in the chain in a suitable position (nodes 4 between 3 and 8, node 15 between 16 and 17). Cycling through all n nodes in the map, the time complexity of Algorithm 1 is $O(n)$.

The aforementioned mapping strategy, i.e. finding a chain of qubits in the coupling map, can be exploited by a noise aware compiler with a slight modification, as described in Ferrari et al. [106]. Specifically, if the found sequence contains more qubits than needed by the circuit, the pass selects a best subset based on two-qubit gates reliability using a sliding window technique. On the other hand, if there are

Algorithm 1 CHAIN($\mathscr{G}$, n)

Input: undirected graph $\mathscr{G}$; number of qubits n used by the circuit
Output: a chain C connecting at least n nodes in $\mathscr{G}$

```
1: 𝒞 ← ∅
2: 𝒮 ← all nodes of 𝒢
3: put 0 into 𝒞
4: 𝒮 ← 𝒮/0
5: ℰ ← 0
6: ℐ ← ∅
7: x = 𝒞[|𝒞| − 1]
8: last_back_step ← −1
9: while |ℰ| < |𝒢| do
10:     𝒩 ← 𝒩_x/E
11:     if |𝒩| ≠ ∅ then
12:         if x + 1 ∈ 𝒩_x then
13:             x ← x + 1
14:         else
15:             x ← min(𝒩_x)
16:         end if
17:         put x into ℰ
18:         put x into 𝒞
19:         𝒮 ← 𝒮/x
20:         if |ℰ| < |𝒢| − 1 then
21:             𝒩 ← ∅
22:             for all q ∈ 𝒩_x do
23:                 if q ∉ ℰ then
24:                     remove=true
25:                     if |N_q| = 1 and |ℰ < |G| − 1| then
26:                         put q into ℰ
27:                         𝒮 ← 𝒮/q
28:                         put q into ℐ
29:                         continue
30:                     end if
31:                     for all r ∈ 𝒩_q do
32:                         if r ∉ ℰ and r = x then
33:                             remove=false
34:                         end if
35:                         if remove=true then
36:                             put q into ℰ
37:                             𝒮 ← 𝒮/q
38:                             put q into ℐ
39:                         end if
40:                     end for
41:                 end if
42:             end for
43:         end if
44:     else
45:         if last_back_step ≠ 𝒞[|𝒞| − 2] and |𝒢| − |ℰ| > |current − 𝒮[0]| then
46:             break
47:         end if
48:         put x into ℐ
49:         𝒞 ← 𝒞/x
50:         x ← 𝒞[|𝒞| − 1]
51:         last_back_step ← x
52:     end if
53: end while
54: if |𝒞| ≥ n then return 𝒞
55: end if
56: CHECKFORISOLATED(𝒢, 𝒞, ℰ, ℐ)
57: EXPANDCHAIN(𝒢, 𝒞, ℐ, n) return 𝒞
```

Algorithm 2 CHECKFORISOLATED($\mathscr{G}$, $\mathscr{C}$, $\mathscr{E}$, $\mathscr{I}$)
Input: an undirected graph $\mathscr{G}$; $\mathscr{C}$ a chain of nodes in $\mathscr{G}$; $\mathscr{E}$ nodes already explored; $\mathscr{I}$ nodes left outside $\mathscr{C}$ during exploration

```
for m = 0 to |G| − 1 do
    if m ∉ E and m ∉ I then
        for all i ∈ I do
            if m ∈ N_i then
                put m into I
                put m into E
                break
            end if
        end for
        for all n ∈ N_m do
            if n ∈ C then
                put m into I
                put m into E
                break
            end if
        end for
    end if
end for
```

Algorithm 3 EXPANDCHAIN($\mathscr{G}$, $\mathscr{C}$, $\mathscr{I}$, n)
Input: an undirected graph $\mathscr{G}$; $\mathscr{C}$ a chain of nodes in $\mathscr{G}$; $\mathscr{I}$ nodes left outside $\mathscr{C}$ during exploration; n the number of qubits used by the circuit

```
r ← (n − |C|)
while r > 0 do
    for all m ∈ I do
        x ← min(N_m ∩ C)
        if x ≠ ∅ then
            put m into C after x
            I ← I / m
            r ← r − 1
            break
        end if
    end for
end while
```

not enough qubits in the sequence, the pass proceeds to insert qubits from the left-over ones, depicted in dark gray in Fig. 2.10, until the necessary number of qubits is reached. These qubits are first scored based on calibration data, and then inserted into the chain after one of their neighboring qubits, starting form the one with the highest score.

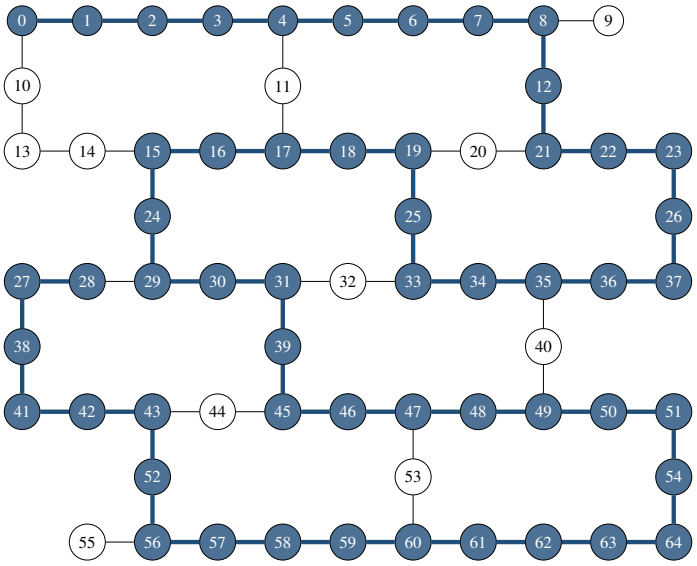


Figure 2.10: Initial mapping on the 65 qubits of the *ibmq_manhattan* device, highlighted in blue.

2.3 Routing

A typical compilation strategy is composed of sequential *passes* that perform placement, routing and optimization tasks. Routing is the task of modifying the circuit in order to move through subsequent mappings, by means of a clever swapping strategy, in order to conform to the qubit layout of the device. From the DQC perspective, this often translates into finding the best scheduling possible for remote operations, either `TeleData` or `TeleGate`, to minimize the EPR pairs consumed.

2.3.1 Local Routing

As previously hinted, current computers are subject to noise, limiting the size of quantum circuits that can be executed reliably. The most advanced quantum compilers are noise-adaptive, i.e., they take noise statistics of the device into account [75, 76, 77, 78], for some or all passes. Noise statistics can be obtained from calibration data, and updated after each device calibration. For example, Qiskit [79] allows for programmatic retrieval of IBM Q devices' calibration data.

In a previous work [106], we designed a heuristic-based routing strategy that accounts for calibration data such as gate reliability and readout errors. The proposed routing pass, denoted as NoiseAdaptiveSwap, assumes a heavy-hexagon lattice for the coupling map of the device, such as the one used by IBM superconducting devices [107]. In heavy-hexagon lattices, the qubits are located on the nodes and edges of each hexagon. Each qubit has either two or three neighbors, meaning the graph has vertices of degree 2 or 3. As a consequence, only three different frequency assignments are necessary for the superconducting qubits, as opposed to a square lattice, which naturally requires at least five different frequencies for addressability. The heavy-hexagon lattice also greatly reduces crosstalk errors since, in principle, only qubits on the edges of the lattice need to be driven by cross-resonance (CR) drive tones [108].

Given an initial mapping of virtual qubits to physical ones, the routing pass, denoted as NoiseAdaptiveSwap, proceeds to compute a *front layer* of non hardware-compliant CNOT gates.

Definition 1. Let C_k be a quantum circuit on k qubits and $U_i(\mathcal{Q}_i)$ a quantum gate acting on a set of qubits $\mathcal{Q}_i$ with $0 < |\mathcal{Q}_i| \leq k$. A layer $\mathscr{L}$ is a set of consecutive gates that can be applied concurrently such that:

1. $\mathcal{Q}_i \cap \mathcal{Q}_j = \emptyset$ for all $U_i, U_j \in \mathscr{L}$, with $i \neq j$
2. $\sum_{U_i \in \mathscr{L}} |\mathcal{Q}_i| \leq k$

Definition 2. The front layer is a layer $\mathscr{F}$ such that $|\mathcal{Q}_i| = 2$ for all $U_i \in \mathscr{F}$.

For convenience, we reformulate circuits by means of the Directed Acyclic Graph (DAG) circuit formalism, as it enables to effectively represent gates dependencies in quantum circuits.

Definition 3. A DAG circuit is a directed acyclic graph where vertices represent gates and directed edges represent qubit dependencies. A directed edge $e_q(i, j)$ between vertices i and j represents a dependency between gate i and gate j with respect to qubit q, i.e., gate i must be executed before j and both gates act on qubit q.

Definition 4. Given a directed graph $G = (\mathscr{V}, \mathscr{E})$, a `Topological order` of G is a linear ordering over vertices in $\mathscr{V}$ such that, for all directed edges $(v, w) \in \mathscr{E}$, v precedes w in the ordering.

Definition 5. Let $v_0 v_1 ... v_n$ be a topological ordering on graph $G = (\mathscr{V}, \mathscr{E})$. Then v_j is a direct topological successor of v_i iff $j > i$ and $\exists e \in \mathscr{E}$ such that $e = (v_i, v_j)$.

To compute the front layer of non hardware-compliant CNOT gates, the pass iterates over all gates in the circuit, in topological order. Gates that do not need routing, such as one-qubit gates or hardware-compliant CNOT gates are added to the set of *executed* gates $\mathscr{X}$. CNOT gates that need to be properly mapped and do not interact with qubits already interested by a CNOT gate in the front layer, are added to the latter. Every gate that is a successor of a CNOT gate in the front layer, is added to the set of *not executed* gates $\bar{\mathscr{X}}$.

From the front layer, the pass computes a list of possible SWAP operations involving at least one qubit interested by a CNOT in the front layer. These SWAP operations are scored with a heuristic cost function $h(s)$, where s denotes the considered SWAP gate. The cost function is computed over all gates in the *front layer* $\mathscr{F}$ plus a set of upcoming gates $\mathscr{U} \subset \bar{\mathscr{X}}$, as shown in Eq. 2.1, where $\pi_s(g_c)$ and $\pi_s(g_t)$ are the physical qubits corresponding to the control and target of gate g with mapping π_s.

$$h(s) = \frac{\alpha}{|\mathscr{F} \cup \mathscr{U}|} \sum_{g \in \mathscr{F} \cup \mathscr{U}} R(\pi_s(g_c), \pi_s(g_t)) + \\ + \frac{1-\alpha}{|\mathscr{F} \cup \mathscr{U}|} \sum_{g \in \mathscr{F} \cup \mathscr{U}} 1 - D(\pi_s(g_c), \pi_s(g_t)) \tag{2.1}$$

Here, R and D are respectively the swap paths reliability matrix and the distance matrix for every qubits pair in the coupling map. Matrix R stores in entry (i, j) the reliability of the most reliable swap path between qubits i and j, where the reliability of a single SWAP along an edge of the coupling map is computed with regard to CNOT gate and readout error rates. Matrix D is the distance matrix and stores at entry (i, j) the shortest distance between qubits i and j. The swap path reliability and the distance between qubits are important components that one would like to maximize and minimize, respectively. The coefficient α gives the opportunity to set equal or opposite weights for them, in the context of the heuristic cost function $h(s)$.

Definition 6. The reliability of SWAP s between qubits i and j is $r(s) = \mu(i, j)^3$, where $\mu(i, j)$ is the success rate of a $CNOT(i, j)$ between qubits i and j, obtained from calibration data.

In the previous definition, it is assumed that the SWAP gate is composed of 3 CNOT gates.

Definition 7. Let $S = s_0 s_1 ... s_k$ be a sequence of SWAP gates (SWAP path) and denote the reliability of SWAP gate s as $r(s)$. Then the reliability of S is given by $\prod_{i=0}^{k} r(s_i)$.

Definition 8. Given a quantum device with n qubits, the SWAP paths reliability matrix $R \in \mathbb{R}^{n \times n}$, is the real matrix that stores in entry (i, j) the maximum reliability or, equivalently, success rate at which qubit i can be moved to a neighbor of qubit j through a sequence of SWAP gates.

Definition 9. Given a quantum device with n qubits, the distance matrix $D \in \mathbb{N}^{n \times n}$, is the matrix that stores in entry (i, j) the minimum distance between qubit i and a neighbor of qubit j.

Both of these matrices can be efficiently precomputed using the Floyd-Warshall algorithm [109]. Matrix R is computed with edge weights corresponding to SWAP reliability, while matrix D has edge weights equal to 1. The SWAP reliability can be easily derived from CNOT calibration data, as each SWAP is usually achieved with three consecutive CNOTs. As matrix D contains entries with incompatible scales with respect to R, both matrices in Eq. 2.1 are normalized.

The first part of Eq. 2.1 sums over the swap reliability of all gates involved in $\mathscr{F} \cup \mathscr{U}$ and divides by $|\mathscr{F} \cup \mathscr{U}|$ to obtain a mean value. The second part follows a similar approach with the distance matrix, except that the sum must be over $1 - D$, as the goal is to maximize the reliability while minimizing the normalized distance.

2.3.2 Remote Gate Scheduling

Worst-Case Scenario

In [65], we analytically derived an upper bound of the overhead induced by quantum circuit compilation for distributed quantum computing:

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

With reference to the last point, we designed a quantum compiler with three key features:

- *general-purpose*, namely, requiring no particular assumptions on the quantum circuits to be compiled;
- *efficient*, namely, exhibiting a polynomial-time computational complexity so that it can successfully compile medium-to-large circuits of practical value;

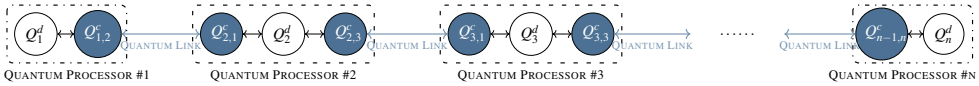


Figure 2.11: Worst-case scenario in terms of overhead induced by the distributed computation: the quantum processors are interconnected through a one-dimensional nearest-neighbor topology, and only one data qubit is available at each quantum processor. Intra-processor coupling between communication qubits omitted for the sake of simplicity.

- *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 considered the worst-case scenario shown in Figure 2.11. More in detail, we assume that only one data qubit is available at each quantum processor[1]. 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 assumed that the quantum processors are interconnected through a one-dimensional nearest-neighbor topology, as shown in Figure 2.11. 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 non-adjacent quantum processors and the maximum distance – in terms of *hops* – between two non-adjacent quantum processors.

[1] 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.

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.

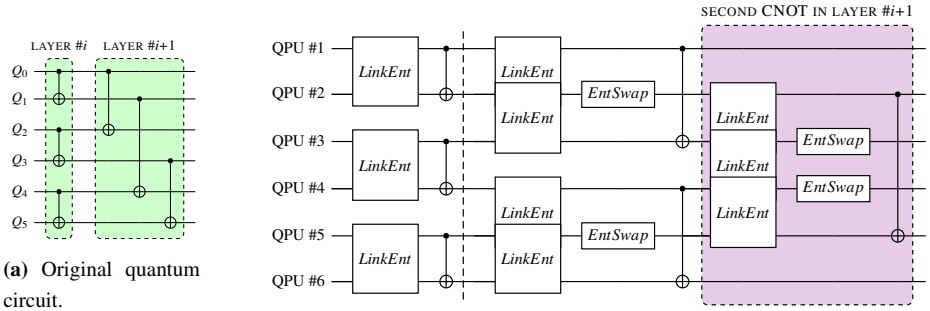
Clearly, we need to choose a metric for measuring the overhead induced by the distributed computation. There exists a general consensus on circuit depth as a key performance metric of circuit compilation. Hence, in the following, we measure the overhead in terms of *number of additional layers required 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.

Let us consider a single layer of the original n-qubit quantum circuit. Clearly, the number of `CNOT`s 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. In the worst-case scenario, each `CNOT` within the quantum circuit – given that it operates on physical qubits stored at different processors – is a remote `CNOT`. As a consequence, the compiler must schedule at most $\frac{n}{2}$ remote `CNOT`s in each layer.

The first strategy for implementing remote `CNOT`s is based on the *entanglement swapping* technique discussed in Section 1.1.3 and shown in Figure 1.11.

Accordingly, each remote `CNOT` is implemented by firstly generating link entanglement [110] among neighbor nodes. To this aim, different techniques for entanglement generation can be employed, depending on the particulars of the underlying qubit technology [36]. Nevertheless, link entanglements can be simultaneously generated, given that each processor is equipped with two communication qubits. Once generated, the entanglement is simultaneously 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 Figure 1.10c. In general, the capability to generate (and to re-generate, 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 inter-connection (shared bus vs. 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.



(a) Original quantum circuit.

(b) Compiled quantum circuit.

Figure 2.12: *Entanglement swapping strategy*. Each remote `CNOT` in Figure 2.12a requires two preliminary tasks: i) *Link Entanglement*, for distributing the entanglement between neighbor nodes, and, ii) the *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 neighbor within the network topology, as for the i-th layer.

The *entanglement swapping based strategy* is outlined in Figure 2.12b 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 Figure 1.11b. 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 [36].

Nevertheless, the figure provides a clear intuition of both: i) the sequentiality constraints between the different tasks, and ii) the parallelism achievable within each task. We note that – for the sake of simplicity – in Figure 2.12b we simply mapped the j-th logical qubit Q_j of layer #i in Figure 2.12a 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.

Whenever the CNOTs *overlaps* within the network topology (as for the CNOTs of the layer #+1 in Figure 2.12a), they must be executed sequentially. Differently, CNOTs that don't overlap (as for the CNOTs of the i-th layer in Figure 2.12a) can be executed simultaneously. 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 Figure 2.12a, 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 CNOT between Q_0 and Q_2 and, hence, they can be performed in parallel.

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:

$$\frac{n}{2} d_{es} \tag{2.2}$$

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:

$$d_{es} = c_{le} + c_{bsm} + c_{cx} \tag{2.3}$$

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 (2.2), we have that the actual depth of an arbitrary d-depth quantum circuit compiled with the *entanglement swapping based strategy* will be always 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. And 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 be always upper-bounded by (2.2).

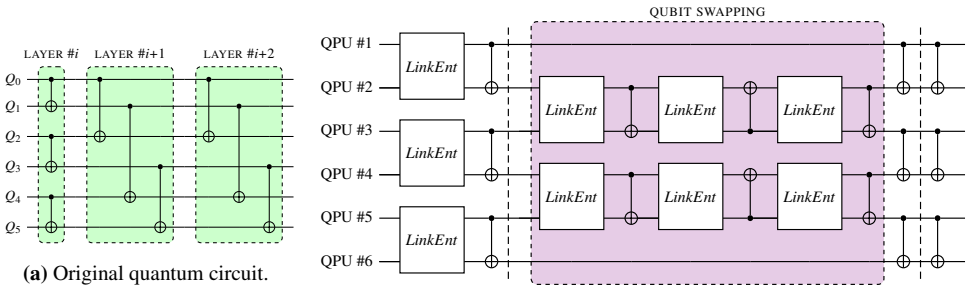
We further note that classical information must be exchanged between the quantum processors. For instance, the entanglement swapping task requires the transmission of classical information (i.e., the measurement output) throughout the quantum network. Hence, in case of long-distance quantum processors, 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 [110], several attempts can be required for establishing link entanglement, and this may impact as well 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.

The *entanglement swapping based strategy* takes full advantage of the augmented connectivity enabled by the communication qubits – as discussed in Section 1.1.3 – 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 Figure 2.13a – a more elaborate strategy – based on *moving* the data qubits – can provide better performance.

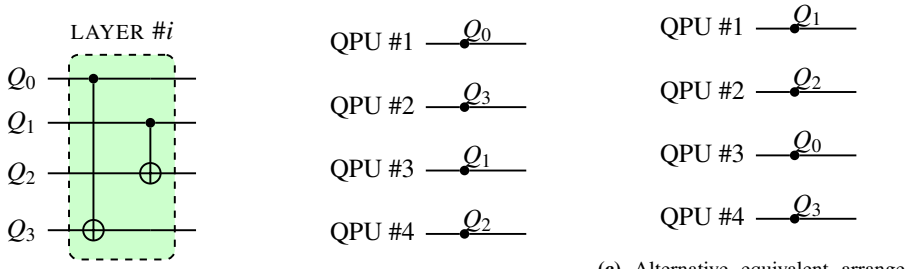
The *data-qubit swapping* strategy is shown in Figure 2.13: its objective is *to arrange* (i.e., to swap) the data qubits within the quantum processors so that eventually each `CNOT` of the original layer operates on qubits stored at neighbor processors



(a) Original quantum circuit.

(b) Compiled quantum circuit.

Figure 2.13: *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 Figure 2.13a. Although not shown in the figure, the entanglement swapping tasks (as in Figure 2.12b) are needed whenever it is necessary to swap data qubits stored at processors that are not neighbor within the network topology.



(a) Original quantum circuit.

(b) Possible arrangement.

(c) Alternative equivalent arrangement.

Figure 2.14: *Data-qubit swapping based strategy*: equivalent mappings. Both Figure 2.14b and Figure 2.14c represent valid arrangements of the data qubits within the remote quantum processors so that each `CNOT` in Figure 2.14a operates on qubits stored at processors neighbor within the network topology.

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

Algorithm 4 Data-Qubit Swapping
Input: n-qubit circuit layer L with mod(n,4) = 0 and $\frac{n}{2}$ CNOTs
Output: layer L with each CNOT operating on neighbor qubits

function SORT(L)
  if $\exists$ CNOT(q_i, q_j) with $i, j \leq \frac{n}{2}$ **then**
    // $\exists$ CNOT(q_k, q_l) with $k, l > \frac{n}{2}$
    SWAP(q_{i+1}, q_j)
    SWAP(q_{k+1}, q_l)
    $L = L \setminus \{q_i, q_{i+1}, q_k, q_{k+1}\}$
  else
    // $\exists$ CNOT$(q_{\frac{n}{2}}, q_l)$ with $l > \frac{n}{2}$
    // and $\exists$ CNOT(q_i, q_{l-1}) with $i < n/2$
    SWAP$(q_{\frac{n}{2}}, q_{l-1})$
    SWAP$(q_i, q_{\frac{n}{2}-1})$
    $L = L \setminus \{q_{\frac{n}{2}-1}, q_{\frac{n}{2}}, q_{l-1}, q_l\}$
  end if
  if $L \neq \emptyset$ **then**
    SORT(L)
  end if
end function

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 [111]:

$$2n - 3 \tag{2.4}$$

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 Figure 2.14. We now formalize these considerations with the following theorem.

Theorem 1. Let us consider the i-th 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:

$$\frac{n}{4} d_{qs} + d'_{qs} \tag{2.5}$$

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

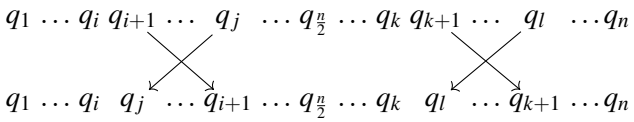
$$d_{qs} = 3\,(c_{le} + c_{bsm} + c_{cx}) \tag{2.6}$$

$$d'_{qs} = c_{le} + c_{cx} \tag{2.7}$$

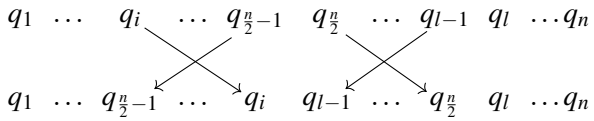
with c_{le} and c_{bsm} denoting the number of layers required to perform the link entanglement task and the entanglement swapping task, respectively, and c_{cx} denoting the number of layers required to perform the remote `CNOT`s once the Bell state has been distributed between two processors.

Proof. The proof easily follows by recognizing that: i) the function $\mathrm{SORT}(\cdot)$, defined in Algorithm 4, is called at most $\frac{n}{4}$ times, and ii) after these calls to $\mathrm{SORT}(\cdot)$, all the `CNOT`s, by acting on qubits stored at neighbor processors, can be executed at once through link entanglement followed by local operations, as shown in Figure 1.10c.
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 $\frac{n}{2}$ `CNOT`s – 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 `CNOT`s do not overlap and two simultaneous `SWAP`

operations can be executed – one in each *half portion* of the original quantum circuit – as shown here:



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 4, 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:



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 (2.5), we have that the actual depth of an arbitrary d-depth quantum circuit compiled with the *data-qubit swapping based strategy* will be always 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 depends also on the repetitions of the same `CNOT` interaction patterns within the original quantum circuit.

Concerning to the number of repetitions of the link entanglement task, in general it depends on the characteristics of the underlying technology through the parameters d_{es}, d_{qs} and d'_{qs} given in equations (2.3), (2.6) and (2.7).

To better clarify this point, let us consider d_{qs}, which inherently denotes the cost for a `SWAP` operation. Having assumed through the manuscript the `CNOT` being the fundamental multi-qubit gate, a single remote `SWAP` operation can be obtained through three remote `CNOT`s. And this is the rationale for the constant factor equal to 3 in (2.6), which accounts for the cost of three remote `CNOT`s.

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 [112], and in such a case d_{qs} is equal to 1. Nevertheless the main result – i.e., equation (2.5) – continues to hold. Furthermore, whenever the `SWAP` gate is the native operation, a single `CNOT` can be obtained through two consecutive `SWAP`s interleaved by single-qubit operations [113]. Hence, the expression of d_{es} must change accordingly but the main result – i.e., equation (2.3) – 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 for every original circuit.

General DQC Architecture

Moving from the worst-case scenario to more general DQC architectures, we designed a compilation pass to schedule remote gates for such architectures and con-

duct a preliminary study on the impact of using both `TeleData` and `TeleGate` operations. The main algorithm is described in Alg. 5, and requires three input items: the quantum circuit to distribute, the configuration of the network onto which such circuit will be executed and a suitable circuit partitioning, as computed by a previous pass in the compilation framework. The pass scans the quantum circuit gate by gate and stops when it encounters a gate that, based on the current partitioning, involves qubits on different QPUs. The pass then searches for feasible `TeleData` operations that could cover[2] either by teleporting one or both qubits on a common QPU. `TeleData` operations are selected by taking into account the memory capacity of each QPU, all the while making sure that no data qubits storing valuable information gets overwritten by a teleportation. Finally, each possible `TeleData` is assigned a cost, which is given by Eq. 2.8:

$$\frac{n_{EPR}}{n_{cov}} delay \tag{2.8}$$

where n_{EPR} is the number of EPR pairs consumed, n_{cov} is the number of covered gates, which may include more than the gate we originally wanted to cover – as shown in Fig. 2.16b – , and $delay$ is the time, measured in discrete intervals, that we need to wait before actually executing the gate. The $delay$ is an estimation based on when the quantum links for entanglement generation were last used and when we would like to execute the gate. It may be the case that before executing a `TeleData` operation, one needs to wait for a previously scheduled one to complete.

The pass selects the `TeleData` operation with the lowest cost and then covers a portion of the remaining uncovered gates[3] with `TeleGate` operations. `TeleGate` operations are chosen and scheduled in a similar manner to the `TeleData` ones. `TeleGates` exploit the cat-entanglement subroutine a shown in Fig. 2.15. `TeleGates` with the cat-entanglement subroutine [87] can be divided in three phases. First, with the cat-entanglement primitive, the one in the blue box in Fig. 2.15, the control qubit of a remote gate is "shared" with the QPU holding the target qubit. Then, gates controlled by the same shared controlled qubit can be be executed locally. Finally, with

[2]By cover we mean to make the gate executable

[3]The dimension of the protion to cover is set by a customizable parameter.

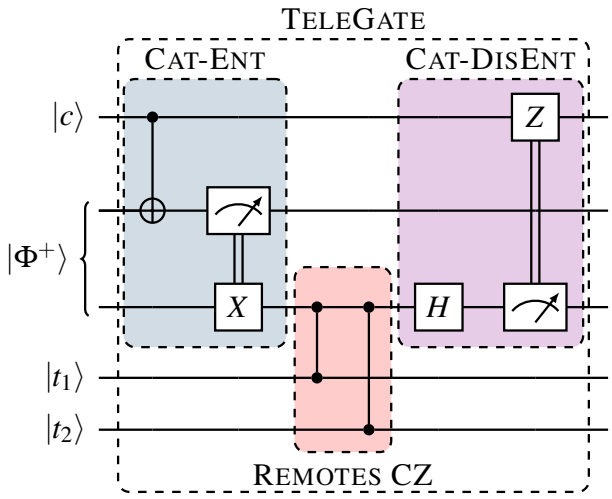


Figure 2.15: Circuit representation of `TeleGate` by means of Cat-Entanglement and Cat-Disentanglement primitives. After the `Cat-Ent` operation, the second half of the entangled pair acts as a shared copy (not an actual copy, due to the no cloning theorem) of the original $|c\rangle$ control qubit. Multiple remote `CZ` with same control qubit and different target can be executed between the `Cat-Ent` and `Cat-DisEnt` operations. In between `Cat-Ent` and `Cat-DisEnt`, the $|c\rangle$ control qubit is entangled with its shared copy and cannot be targeted by other gates.

Algorithm 5 Remote Gates Scheduler

Input: quantum circuit QC, network configuration N and circuit partitioning P

Output: quantum circuit with remote gates D

```
1:  function SCHEDULE
2:      D ← ∅
3:      covered ← ∅
4:      for all g ∈ QC do
5:          if g ∉ D then
6:              if g is local then
7:                  put g into covered and D
8:              else
9:                  teledata ← FINDTELEDATA(g, N, P)
10:                 telegate ← FINDTELEGATE(g, N, P)
11:                 if COST(teledata) < COST(telegate) then
12:                     put teledata into D
13:                 else
14:                     put telegate into D
15:                 end if
16:                 put g into covered and D
17:                 put extra covered gates into D and covered
18:             end if
19:         end if
20:     end for
21: end function
```

the cat-disentanglement primitive, the one in the violet box in Fig. 2.15, the shared state of the control qubit is restored at the first QPU.

Both `TeleData` and `TeleGate` can either migrate one qubit to the other's QPU or both to a different one, as shown in Fig. 2.16, depending on the cost of such operation, computed as in Eq. 2.8. Fig. 2.16a shows the first case, in which gate g_0 is covered by sharing qubit q_1 with QPU_1 through one `TeleGate` operation. Fig. 2.16b

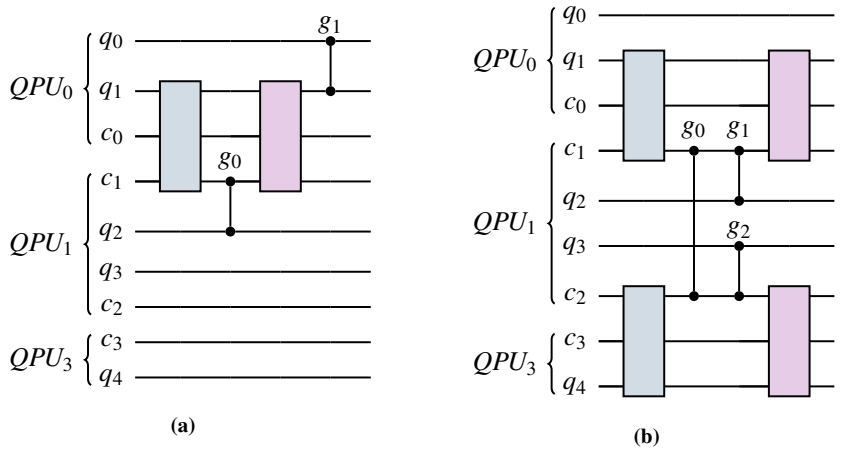


Figure 2.16: Gates can be covered by either migrating one qubit to the other qubit's QPU or both to a different one. This concept is valid for both `TeleGate` and `TeleData` operations. **(a)** Gate g_0 is covered by sharing qubit q_1 with QPU_1 using one `TeleGate`. **(b)** Qubits q_1 and q_4 are shared with QPU_2 using two `TeleGatess`, gates g_0, g_1 and g_2 are consequently covered.

depicts the second case, where gate g_0 is covered by sharing qubits q_1 and q_4 with QPU_1, using two `TeleGates`. By doing this, also gates g_1 and g_2 are covered. The same concept can be applied with `TeleData` operations.

The pass also compiles the same portion of circuit but by scheduling only `TeleGate` operations. Finally, the pass can compute a cost for the portions of the circuit – one with `TeleData` and `Telegate`, the other with just `TeleGate` – and select the strategy with the lowest cost. This time cost is simply the amount of consumed EPR pairs. Finally, the pass resumes scanning gates in search of the next gate to cover.

2.4 Combining Local Routing And Remote Gate Scheduling

As mentioned at the beginning of this chapter, we want to propose a framework that can bridge the gap between compilation for local and distributed quantum computing. To this aim, in this Section we design a simple local routing pass that takes as

input a partitioned circuit with already scheduled remote operations and handles the local routing accordingly. The pass requires the partitioned circuit, with scheduled remote operations, the network configuration, and each QPU configuration, specifically coupling maps including data qubits to communication qubits connections.

The core strategy is straightforward. The pass scans the circuit and for every gate that involves qubits not directly connected on their specific QPU, computes the shortest sequence of necessary `SWAP` gates. When it encounters a `TeleData` or `TeleGate` operations, it first checks if the involved data qubits are in proximity of one of the available communication qubits, among those corresponding to the quantum links used by the remote operation. If not, it computes the shortest paths to the less recently used communication qubit. We choose the less recently used communication qubit to avoid as much delay as possible in entanglement generation. At this stage of compilation, due to local `SWAP`s, the state of a data qubit may now reside on a communication qubit and vice versa. This is not necessarily an issue [114], but we need to move the communication qubit back to its original position, after the remote operation is completed and before it is used again. This is crucial, as we do not want to loose the state of a data qubit physically stored at a communication qubit, due to a new remote operation. An example of such instance in shown in Fig. 2.17.

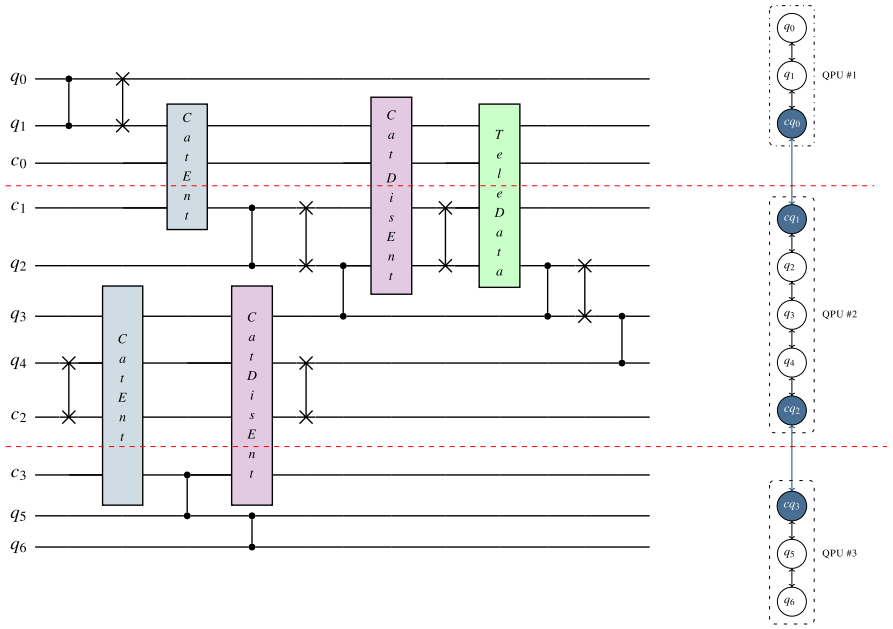


Figure 2.17: Example of remote gate scheduling and local routing. Local gates are interlaced with `Cat-Ent`, `Cat-DisEnt` and `TeleData` operations as well as `SWAP` gates.

Chapter 3

Implementation

3.1 Local Mapping

We implemented the algorithms proposed in [96] in a quantum compiler written in Python language, denoted as PADQC[1]. In the PADQC framework, quantum circuits are represented by QCircuit objects, which are based on the well known formalism of Directed Acyclic Graphs (DAGs). In DAGs, nodes represent quantum gates and the edges connecting them correspond to qubits and bits. A QCircuit can then be easily converted into Open QASM [70] and vice versa, allowing PADQC to interact with Qiskit.

3.2 Local Routing

A Python implementation of the noise-adaptive compiler presented in Sec. 1.2.1 is also available on GitHub.[2] It has been designed as a Qiskit pass [107], thus it can be used with any quantum device supported by Qiskit.

[1] Source code: `https://github.com/qis-unipr/padqc`

[2] Source code: `https://github.com/qis-unipr/noise-adaptive-compiler`

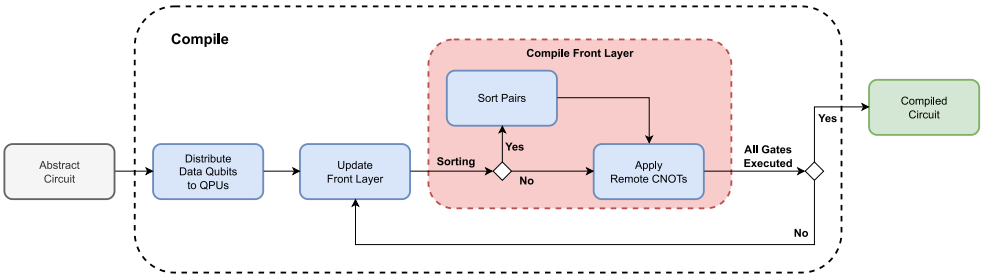


Figure 3.1: Workflow of the proposed compiler.

3.3 Remote Gate Scheduling

We implemented the strategies discussed in Section 2.3.2 in Python, using Qiskit [79] 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 depicted in Fig. 2.11. To model the worst-case scenario, 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 by Algorithm 7 and illustrated in Fig. 3.1. 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 with Algorithm 8. 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 6, where one can choose between the two strategies discussed in Section 2.3.2. When the *entanglement swapping based strategy* is adopted, remote `CNOT` gates preceded by entanglement swapping are ap-

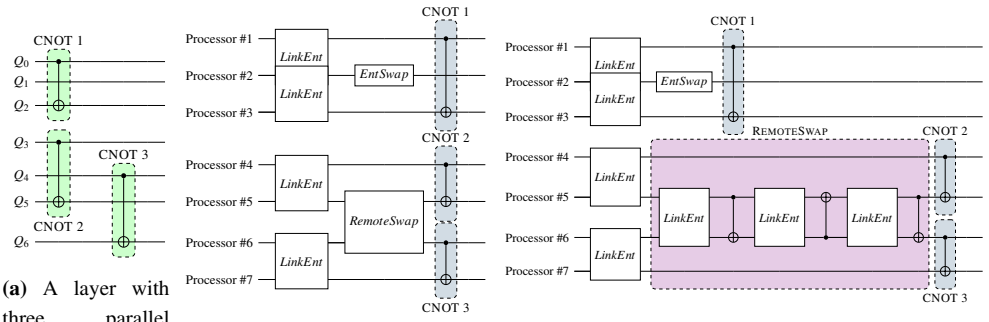
plied whenever the involved QPUs are not neighbors. Note that to perform entanglement swapping, as well as remote CNOTs, we need to generate link entanglement between all involved QPUs.

Regarding the more advanced *data-qubit swapping based strategy*, the list representing the interaction between qubits is prepared and then Algorithm 9 is used to compute the data swap operations needed to reorder the qubits. Referring to Fig. 3.2a, the list representing the interaction between qubits before swapping would be $[112323]$, while the sorted list 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. 2.11, 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. 3.2a, it may be still necessary to perform some entanglement swapping operations, as shown in Fig. 3.2b. 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 an hybrid between the two strategies described in Sections 2.3.2, avoiding data-qubit swapping if not necessary and resorting instead to entanglement swapping.

As shown in Fig. 3.2, starting from the layer in Fig. 3.2a, following the *data-qubit swapping based strategy*, in Fig. 3.2b we apply a remote SWAP between qubit 3 and qubit 4, and then we can execute all remote CNOTs in parallel.



(a) A layer with three parallel CNOTs.

(b) The layer distributed with the Sort strategy.

(c) The distributed layer after decomposing the Remote SWAP.

Figure 3.2: Compilation of a layer with three parallel CNOTs using the sorting strategy.

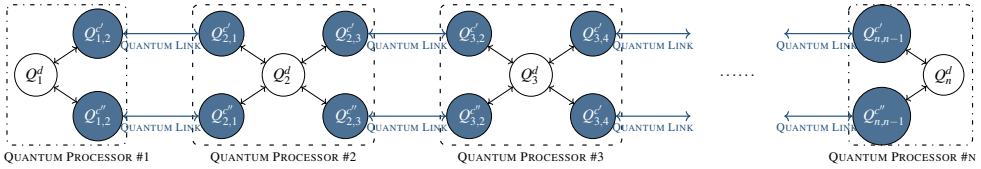


Figure 3.3: Improving over the worst-case scenario topology. Only one data qubit is available at each quantum processor, but neighboring quantum processors are connected with two quantum links, realized by 2 EPR pairs.

Moving from the topology illustrated in Fig. 2.11 to the one in Fig. 3.3, we devised a different strategy to execute swaps by exploiting the augmented connectivity, described in Algorithm 6. 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 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 layer 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, Alg. 7 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. Alg. 6 loops through all necessary swaps found by Alg. 9, and applies remote `CNOT`s. 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 the we need at most $n/4$ swaps, the computational complexity of Alg. 6 turns out to be $O(n^2)$. The overall computational complexity of distributing a circuit is therefore $O(dn^2)$.

Algorithm 6 COMPILEFRONTLAYER

Input: the front layer $\mathscr{F}$ to be compiled, the executed gates $\mathscr{E}$ until now
Output: the executed gates $\mathscr{E}$ updated with the compiled front layer $\mathscr{F}$

if *data-qubit swapping based strategy* **then**
  prepare *interactions* vector
  $swaps \leftarrow$ SORTPAIRS(INTERACTIONS)
  for all $swap \in swaps$ **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 $gate \in \mathscr{F}$ **do**
  REMOTECNOT($gate.control, gate.target$)
end for
return $\mathscr{E}$

function REMOTECNOT($control, target$)
  if *control* and *target* are not on neighboring QPUs **then**
    $qpu_0 \leftarrow$ QPUs of *control*
    $qpu_n \leftarrow$ QPUs of *target*
    for $i \in \{0, .., n-2\}$ **do**
      add *link entanglement* between qpu_i and qpu_{i+1} to $\mathscr{E}$
    end for
    add *entanglement swap* between qpu_0 and qpu_n to $\mathscr{E}$
  end if
  add *link entanglement* between qpu_0 and qpu_n to $\mathscr{E}$
  add a remote CNOT between *control* and *target* to $\mathscr{E}$
end function

function TELEPORT($q1, q2$)
  if $q1$ and $q2$ are not on neighboring QPUs **then**
    $qpu_0 \leftarrow$ QPUs of $q1$
    $qpu_n \leftarrow$ QPUs of $q2$
    for $i \in \{0, .., n-2\}$ **do**
      add *link entanglement* between qpu_i and qpu_{i+1} to $\mathscr{E}$ using two available EPR pairs between QPUs
    end for
    $e_{01}, e_{02} \leftarrow$ the two communication qubits at qpu_0
    $e_{n1}, e_{n2} \leftarrow$ the two communication qubits at qpu_n
    add *entanglement swap* between e_{01} at qpu_0 and e_{n1} at qpu_n to $\mathscr{E}$
    add *entanglement swap* between e_{02} at qpu_0 and e_{n2} at qpu_n to $\mathscr{E}$
    add *quantum teleportation* between $q1$ and e_{n1} to $\mathscr{E}$
    add *quantum teleportation* between $q2$ and e_{02} to $\mathscr{E}$
    add local *swap* between $q1$ and e_{02}
    add local *swap* between $q2$ and e_{n1}
  end if
end function

Algorithm 7 COMPILE

Input: the *circuit* to be compiled

Output: the compiled circuit

$\mathscr{G} \leftarrow$ all gates from *circuit*
$\mathscr{E} \leftarrow \emptyset$ // executed gates
create *new_circuit* as an empty circuit
while $\mathscr{G} \neq \emptyset$ **do**
  $\mathscr{G}, \mathscr{F}, \mathscr{E} \leftarrow$ UPDATEFRONTLAYER($\mathscr{G}, \mathscr{E}$) // $\mathscr{F}$ is the front layer
  if $\mathscr{F} \neq \emptyset$ **then**
    $\mathscr{E} \leftarrow$ COMPILEFRONTLAYER($\mathscr{F}, \mathscr{E}$)
  end if
end while
for all $gate \in \mathscr{E}$ **do**:
  add $gate$ to *new_circuit*
end for
return *new_circuit*

Algorithm 8 UPDATEFRONTLAYER

Input: $\mathscr{G}$ gates to be executed, $\mathscr{E}$ executed gates until now
Output: updated $\mathscr{G}$, updated $\mathscr{E}$, new front layer $\mathscr{F}$

$\mathscr{A} \leftarrow \emptyset$ // allocated qubits
$\mathscr{F} \leftarrow \emptyset$ // new front layer
$\mathscr{R} \leftarrow \emptyset$ // gates to remove
$width \leftarrow$ total number of data qubits available
for all $gate \in \mathscr{G}$ **do**
  if $|\mathscr{A}| = width$ **then**
    break
  end if
  if $\mathscr{A} \cap gate.qubits = \emptyset$ **then**
    if $gate$ is a one-qubit gate **then**
      add $gate$ to $\mathscr{E}$
    else
      add $gate$ to $\mathscr{F}$
      add $gate$ to $\mathscr{E}$
      add $gate.qubits$ to $\mathscr{A}$
    end if
    add $gate$ to $\mathscr{R}$
  else:
    add $gate.qubits$ to $\mathscr{A}$
  end if
end for
for all $gate \in \mathscr{R}$ **do**
  remove $gate$ from $\mathscr{G}$
end for
return $\mathscr{G}, \mathscr{F}, \mathscr{E}$

Algorithm 9 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) $\bmod 4 \neq 0$ **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)
$mask1 \leftarrow \{0,...,\frac{n}{2}-1\}$
$mask2 \leftarrow \{\frac{n}{2},...,n-1\}$
$cycle \leftarrow 0$
while $cycle < n/4$ **do**
    $w \leftarrow$ indexes of unique values in v
    $index\_last \leftarrow \{0,...,|v[mask1]|\} \setminus w$
    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.*** $\{0,...,|v[mask]|\} \setminus w$
    $index\_first \leftarrow$ ***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

3.4 Combining Local Routing And Remote Gate Scheduling

We also implemented the framework presented in Chapter. 2. To partition quantum circuits we used the Tabu search approach suggested by Sundaram et al. [89]. For the local mapping we used Qiskit's DENSELAYOUT pass and for the remote gate scheduling we followed the strategy proposed in Sec. 2.3.2. Finally we combined remote gate scheduling and local routing with a modified version of Qiskit's BASICSWAP. This customized routing pass follows the strategy explained in Sec. 2.4 to take into account each QPU's local configuration and the remote gates schedule.

Chapter 4

Evaluation

4.1 Local Mapping

Using an Intel Xeon E5-2683v4 2.1GHz with 50 GB of RAM, we benchmarked the quantum compiler presented in the first part of Section 2.2.1, denoted as PADQC, with different quantum circuits. In particular, we considered a few quantum chemistry circuits for the RyRz heuristic [98] wavefunction Ansätz (Fig. 2.4), and an heterogeneous set of quantum circuits that has been used in most reference works [85, 83]. We assumed IBM Quantum hardware, namely *ibmq_tokyo* and *ibmq_almaden* architectures. They both have 20 qubits, but their coupling maps are quite different (Fig. 2.7).

			CNOT Gate Count					
			ibmq_tokyo			*ibmq_almaden*		
Circuit Name	n	Initial	Qiskit(SABRE)	PADQC+Qiskit(SABRE)	t|ket⟩	Qiskit(SABRE)	PADQC+Qiskit(SABRE)	t|ket⟩
H2_RYRZ	4	30	30	21	30	60	21	70
LiH_RYRZ	12	330	788	101	898	1256	101	1260
H2O_RYRZ	14	455	1242	121	1301	1763	121	1738
Random_20q_RYRZ	20	950	2735	182	3053	3976	201	4146

Table 4.1: Circuit CNOT gate count of the compiled quantum chemistry circuits, considering the *ibmq_tokyo* and *ibmq_almaden* architectures.

			CNOT Depth					
			ibmq_tokyo			*ibmq_almaden*		
Circuit Name	n	Initial	Qiskit(SABRE)	PADQC+Qiskit(SABRE)	t|ket⟩	Qiskit(SABRE)	PADQC+Qiskit(SABRE)	t|ket⟩
H2_RYRZ	4	21	21	21	21	60	21	70
LiH_RYRZ	12	69	488	101	575	648	101	703
H2O_RYRZ	14	81	675	121	813	846	121	875
Random_20q_RYRZ	20	117	1361	182	1648	1562	201	1629

Table 4.2: Circuit CNOT depth of the compiled quantum chemistry circuits, considering the *ibmq_tokyo* and *ibmq_almaden* architectures.

We compared Qiskit(SABRE) with t|ket⟩ and Qiskit(SABRE) preceded by PADQC transformations and mapping. Given that on NISQ devices multi-qubit operations tend to have error rates and execution times an order of magnitude worse than single-qubit ones [115], the performance indicators used are CNOT gate count and CNOT depth of the circuit, i.e., the depth of the circuit taking into account only CNOT gates. As shown in Table 4.1 and Table 4.2, the combination of PADQC and Qiskit(SABRE) outperforms both Qiskit(SABRE) alone and t|ket⟩ on all RyRz circuits tested, regardless of the considered architecture.

We also tested PADQC on a larger set of circuits. We compiled a total of 190 circuits[1], taken from RevLib [116], Quipper [117] and Scaffold [118], plus the quantum chemistry ones. The results are summarized in Fig. 4.1 and Fig. 4.2, where the initial value of the considered metric is on the x-axis and the value for the compiled circuit on the y-axis.

When compiling on the *ibmq_tokyo* architecture, we can see, with the help of tendency lines calculated using the least squares method, that the combination of PADQC and Qiskit(SABRE) not only boosts Qiskit(SABRE) performance but can also compete with t|ket⟩. As we switch to the less connected *ibmq_almaden* architecture, PADQC can still boost Qiskit(SABRE) performance while being up to par with t|ket⟩.

[1]Benchmark circuits QASM files: `https://github.com/qis-unipr/padqc/tree/master/benchmarks_qasm`

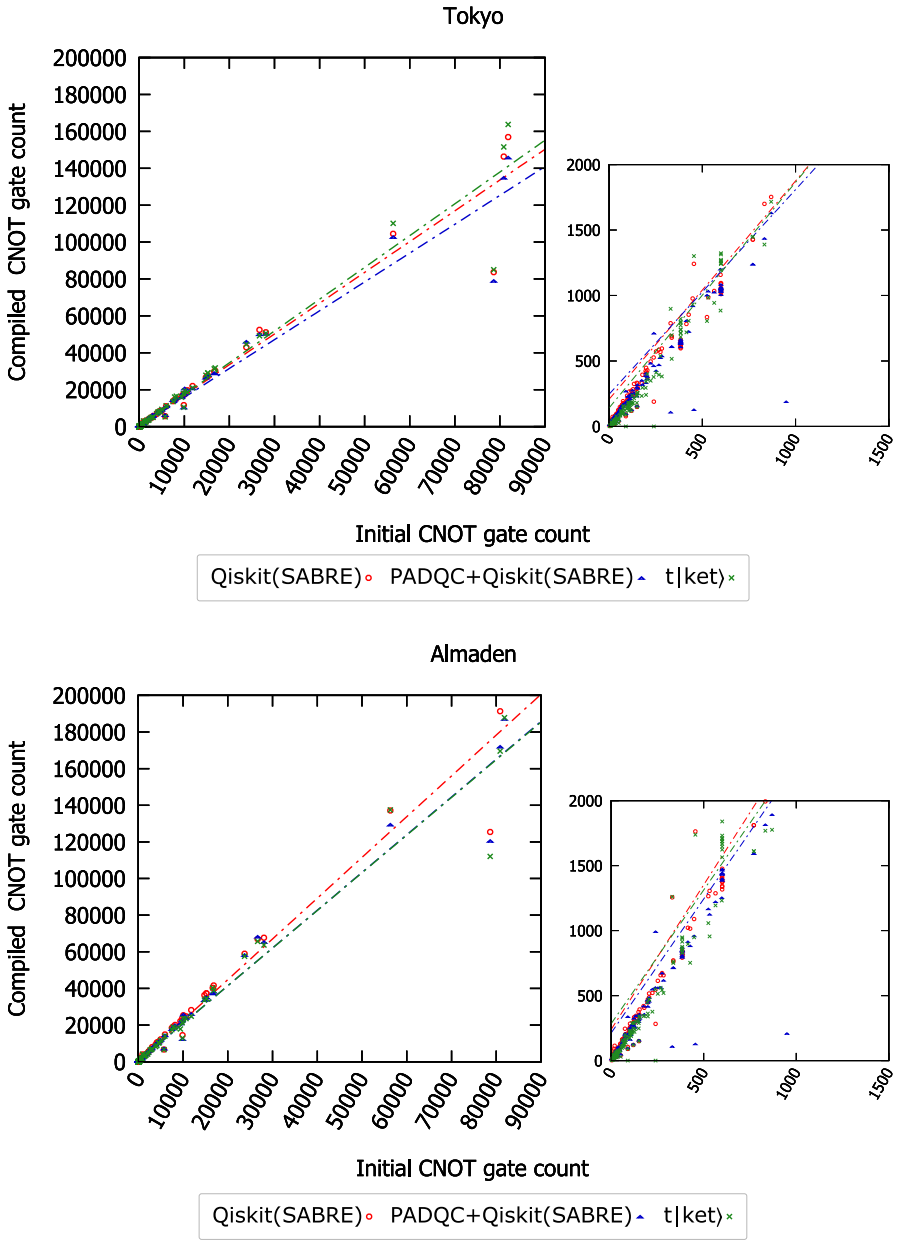


Figure 4.1: Comparing the CNOT gate count of the circuits compiled on *ibmq_tokyo* and *ibmq_almaden*.

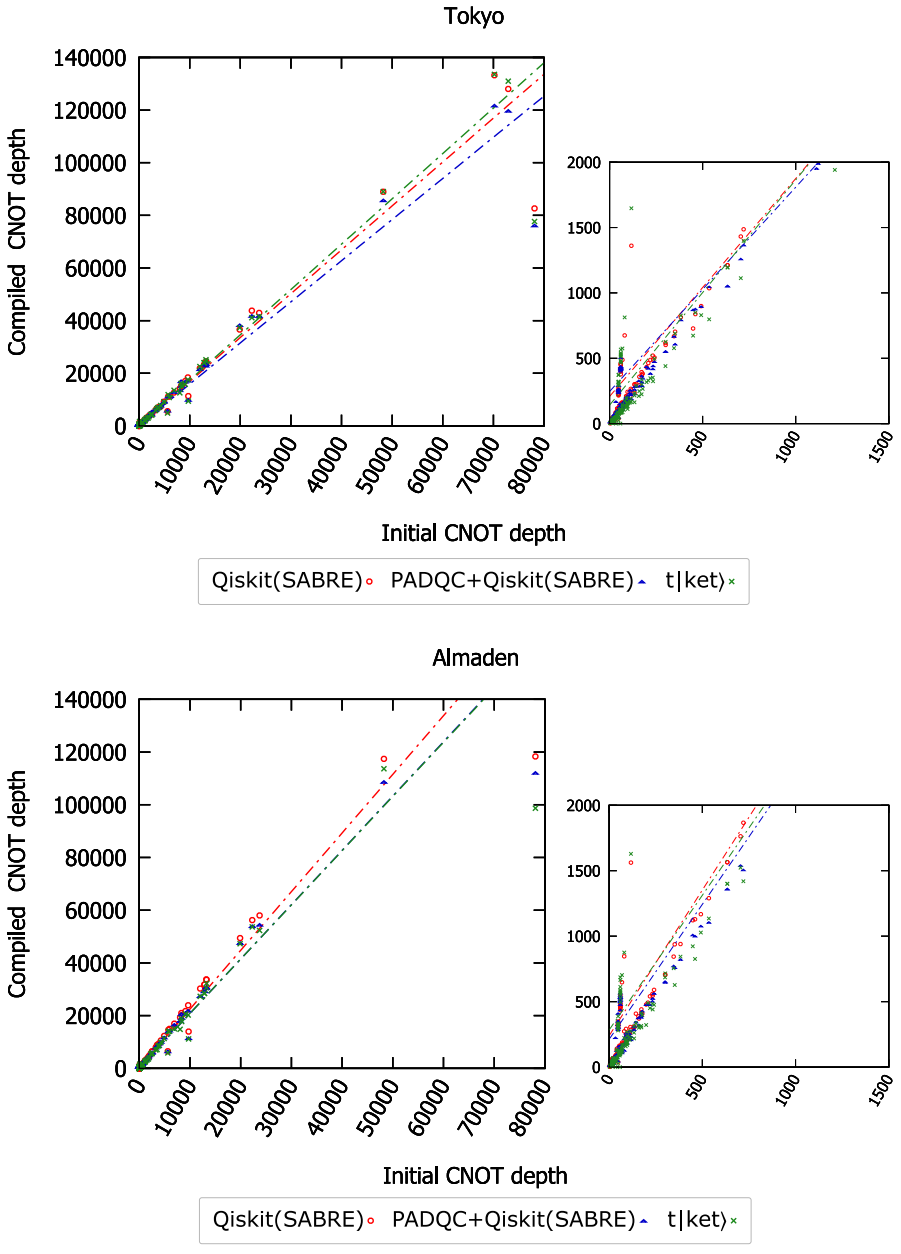


Figure 4.2: Comparing the CNOT depth of the circuits compiled on *ibmq_tokyo* and *ibmq_almaden*.

4.2 Local Routing

We evaluated the proposed noise-adaptive compilation strategy, compared with some of the most advanced state-of-art approaches, over the circuit classes presented in [81][2], with 200 circuits for each class and different number of qubits. The circuits are divided into three classes. Fig. 4.3 shows the distributions of the number of gates in the considered circuits, for each class and number of involved qubits.

(a) Deep circuits. **(b)** Square circuits **(c)** Shallow circuits

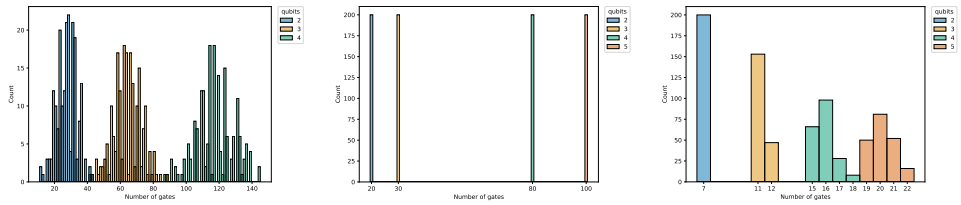


Figure 4.3: Distributions of the number of gates for each class of circuit, depending on the number of involved qubits. By construction, all the square circuits have the same number of gates, which increases with the number of involved qubits.

For each circuit the ideal output distribution p_C was obtained using Qiskit's statevector simulator. The same simulator was used to sample noisy circuits from the noise model obtained with calibration data of the IBM *ibmq_casablanca* 7-qubit device.

We tested the following compilers:

- *qiskit*, Qiskit's standard compiler with optimization level 3;
- *qiskit_noise*, Qiskit's compiler with NoiseAdaptiveLayout and optimization level 3;
- *pytket*, the t|ket⟩ compiler with NoiseAwarePlacement and maximum optimization level;

[2] https://doi.org/10.5281/zenodo.3832121

- *na*, the proposed noise-adaptive placement and routing passes integrated with Qiskit and optimization level 3;
- *qiskit_na*, the proposed routing pass combined with Qiskit's NoiseAdaptive-Layout and optimization level 3;
- *t* denotes the application of CNOT cascade transformations, as described in [96].

For each of the above we used Qiskit v0.23.6 and t|ket⟩ v0.7.2. We remark that the *pytket_na* compiler was not tested, as the integration of *pytket* and *na* is possible only with regards to swap passes, not for the placement ones.

The results are reported in Fig.4.4, Fig.4.5 and Fig.4.6. For each compiler configuration, we used box plots to depict the distributions of the resulting values for the following figures of merit, as introduced in [81]. Let us denote an n qubit circuit as C, the ideal output distribution of C as p_C and the output distribution produced by the compiled implementation of C as D_C.

- **Hellinger fidelity** - The Hellinger fidelity between D_C and p_C is

$$F_C = \left(\sum_{x \in \{0,1\}^n} \sqrt{D_C(x) p_C(x)} \right)^2 . \tag{4.1}$$

  We would like that $F_C = 1$.

- **Heavy Output Generation (HOG)** - An output $z \in \{0,1\}^n$ is heavy for a quantum circuit C, if $p_C(z)$ is greater than the median of the set $\{p_C(x) : x \in \{0,1\}^n\}$. The HOG probability of D_C, i.e., the probability that samples drawn from from D_C will be heavy outputs in p_C, is

$$\mathrm{HOG}(D_C, p_C) = \sum_{x \in \{0,1\}^n} D_C(x) \delta_C(x) \tag{4.2}$$

  where $\delta_c(x) = 1$ if x is heavy for C, and $\delta_c(x) = 0$ otherwise. We would like $\mathrm{HOG}(D_C, p_C) > 1/2$, as it would help us distinguish between a good implementation of C and an attempt to mimic it by generating random bitstrings. Of course this is true if p_C is sufficiently far from uniform.

- l_1**-norm distance** - The l_1-norm distance between D_C and p_C is

$$l_1(D_C, p_C) = \sum_{x \in \{0,1\}^n} |D_C(x) - p_C(x)|. \tag{4.3}$$

  We would like that $l_1(D_C, p_C) = 0$.

- **CNOT Count** - In a quantum circuit, the number of CNOT gates is denoted as CNOT count.

- **CNOT Depth** - In a quantum circuit, the number of layers containing CNOT gates is denoted as CNOT depth.

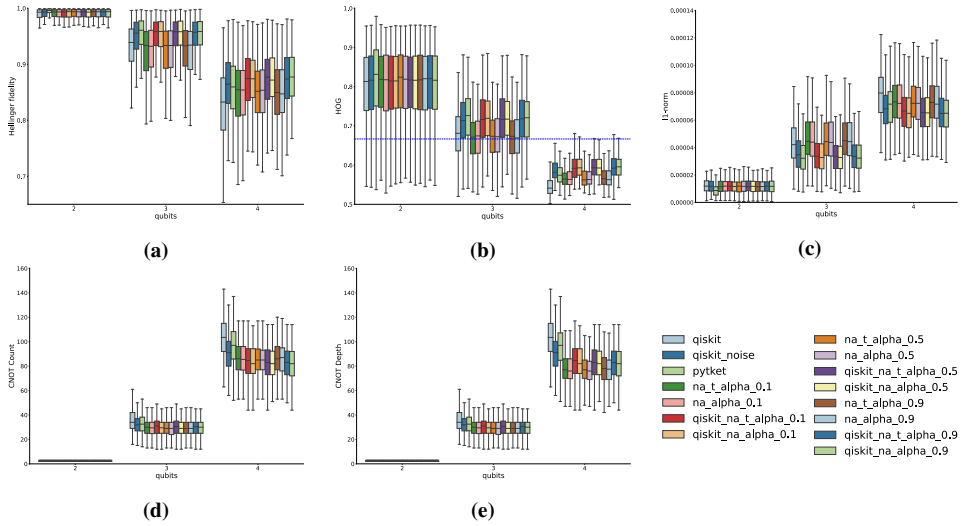


Figure 4.4: Comparison of compilation strategies with deep circuits using (4.4a) Hellinger fidelity, (4.4b) HOG, (4.4c) l_1-norm, (4.4d) CNOT count and (4.4e) CNOT depth as metrics.

The reader may observe that, for deep and square circuits, the best results in Hellinger fidelity, HOG, and l_1-norm are achieved with *qiskit_na*. Interestingly, with this compiler, the CNOT depth is worse than with the other compilers. Indeed, improving the quality of a quantum computation with a noise-adaptive compilation

strategy does not forcedly imply reducing the depth of the circuit. Regarding shallow circuits, *qiskit_na* and *pytket* have the same performance.

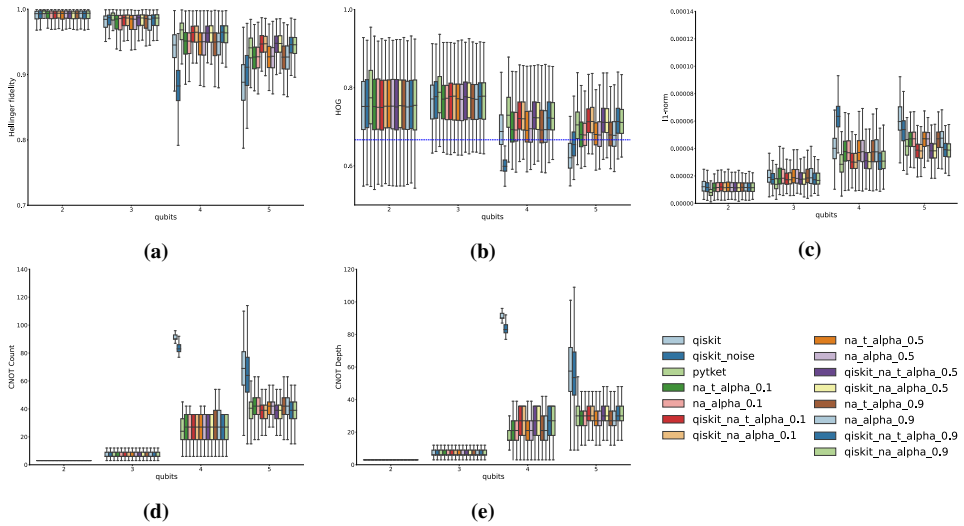


Figure 4.5: Comparison of compilation strategies with square circuits using (4.5a) Hellinger fidelity, (4.5b) HOG, (4.5c) l_1-norm, (4.5d) CNOT count and (4.5e) CNOT depth as metrics.

Regarding the impact of α, one may observe that, with deep circuits, $\alpha = 0.9$ produces better Hellinger fidelity. Instead, with square and shallow circuits, $\alpha = 0.5$ seems a better choice. The difference is nevertheless minimal. This could be due to the small number of qubits in the circuits and in the considered device.

4.3 Remote Gate Scheduling

We compared the compilation for the worst-case scenario, presented in Section 2.3.2 with the one proposed by Andrés-Martínez *et al.* [86], 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 Figure 4.7b, where one can clearly see the connectivity disparity, compared

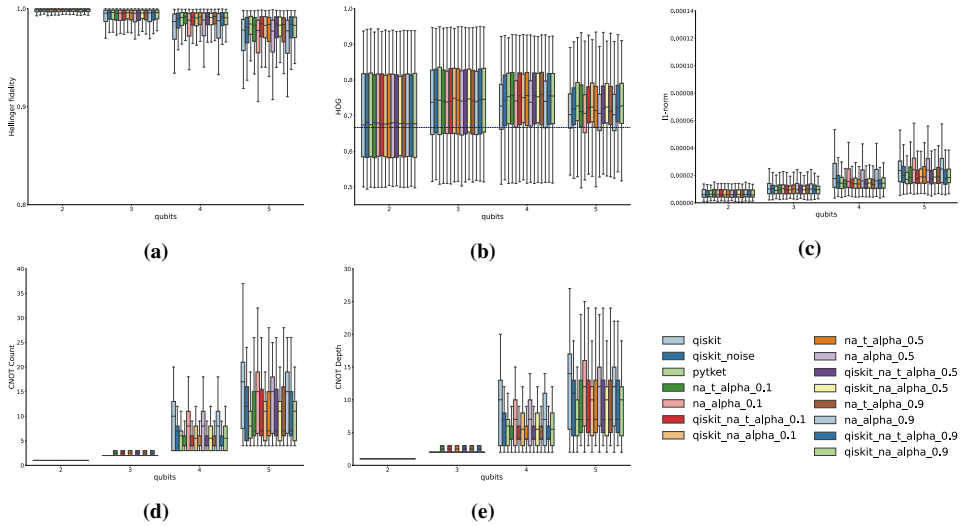
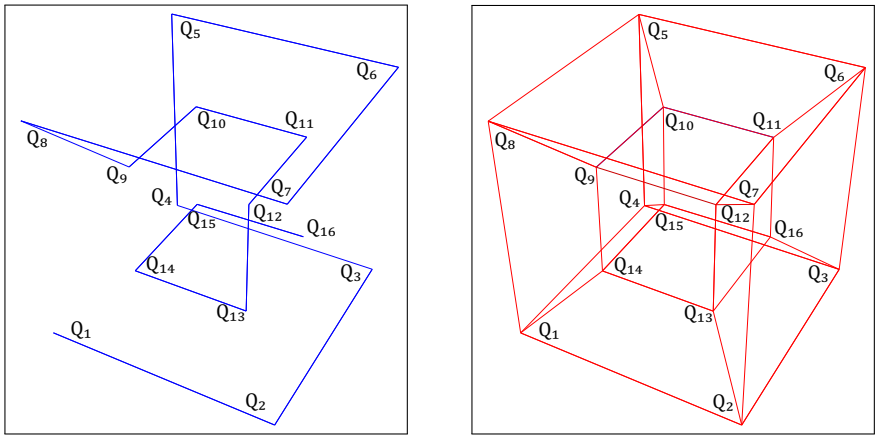


Figure 4.6: Comparison of compilation strategies with shallow circuits using (4.6a) Hellinger fidelity, (4.6b) HOG, (4.6c) l_1-norm, (4.6d) CNOT count and (4.6e) CNOT depth as metrics.

to the worst-case topology illustrated in Figure 4.7a. In Figures 4.8-4.10, 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 data set, for all compiling strategies including the state of the art (such data have been included in Table 4.3).

Figure 4.8 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. Figure 4.8c also shows that choosing the *data-qubit swapping based strategy* against the *entanglement swapping based strategy* is generally the best choice.

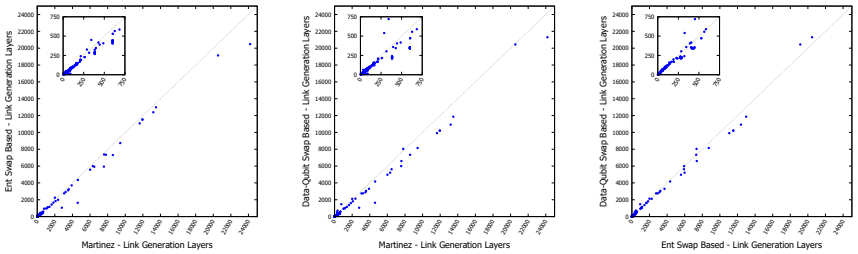
Regarding the depth of the distributed circuits, illustrated in Figure 4.9, we can see that for some circuits our compiler clearly outperforms Andrés-Martínez's one.



(a) Linear Nearest Neighbor topology.

(b) Hypercube topology.

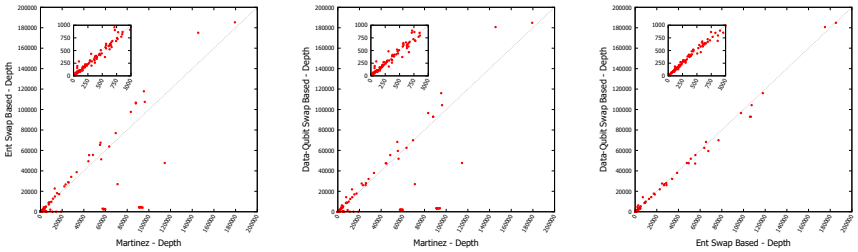
Figure 4.7: Comparison of the worst-case scenario linear nearest neighbor topology with the hypercube topology assumed by the compiler proposed by Andrés-Martínez *et al.* [86]. As shown in Figure 4.7a, for $n = 16$ QPUs the longest path between two QPUs consists of $n - 1 = 15$ links, while with the hypercube topology in Figure 4.7b is only about $log_2 n = 3$ links.



(a) **(b)** **(c)**

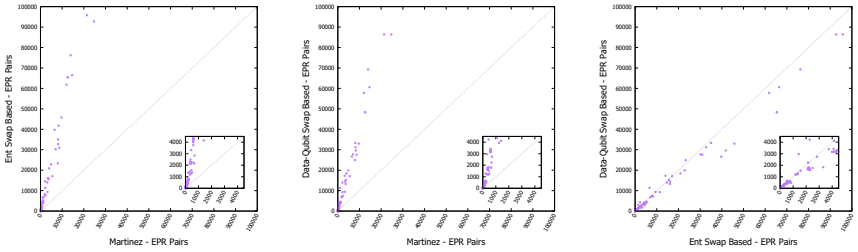
Figure 4.8: Comparing the *entanglement swapping based* and *data-qubit swapping based* strategies of our compiler with Andrés-Martínez's compiler [86], in terms of link entanglement generation layers. Our compiler distributes circuits on the worst-case topology, with only one link between neighboring QPUs (illustrated in Figure 2.11), while Andrés-Martínez's one exploits a more favorable topology, i.e., the hypercube illustrated in Figure 4.7b. Both topologies are characterized by one data qubit per QPU.

Figure 4.9c confirms that the *data-qubit swapping based strategy* is better than the *entanglement swapping based* one.



(a) **(b)** **(c)**

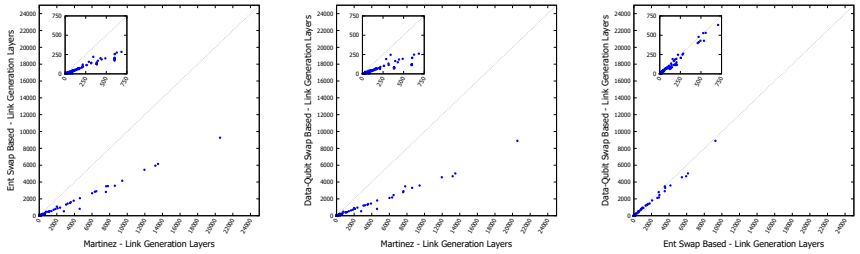
Figure 4.9: Comparing the *entanglement swapping based* and *data-qubit swapping based* strategies of our compiler with Andrés-Martínez's compiler [86], in terms of circuit depth. Our compiler distributes circuits on the worst-case topology, with only one link between neighboring QPUs (illustrated in Figure 2.11), while Andrés-Martínez's one exploits a more favorable topology, i.e., the hypercube illustrated in Figure 4.7b. Both topologies are characterized by one data qubit per QPU.



(a) **(b)** **(c)**

Figure 4.10: Comparing the *entanglement swapping based* and *data-qubit swapping based* strategies of our compiler with Andrés-Martínez's compiler [86], in terms of consumed EPR pairs. Our compiler distributes circuits on the worst-case topology, with only one link between neighboring QPUs (illustrated in Figure 2.11), while Andrés-Martínez's one exploits a more favorable topology, i.e., the hypercube illustrated in Figure 4.7b. Both topologies are characterized by one data qubit per QPU.

With respect to the number of generated Bell states, depicted in Figure 4.10, it



(a) **(b)** **(c)**

Figure 4.11: Comparing the *entanglement swapping based* and *data-qubit swapping based* strategies of our compiler with Andrés-Martínez's compiler [86] over link entanglement generation layers. Our compiler distributed circuits on the topology illustrated in Figure 4.12, with two links between neighboring QPUs(illustrated in Figure 2.11), while Andrés-Martínez's one exploits a more favorable topology, i.e., the hypercube illustrated in Figure 4.7b. Both topologies are characterized by one data qubit per QPU.

can be observed that our compiler consumes a fair amount of Bell states compared to Andrés-Martínez's one. This was expected and it is mostly due to the fact that Andrés-Martínez's compiler benefits from a hypercube network topology, as shown in Figure 4.7b. 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 Figure 4.7a. In our linear topology, to generate a link between two non neighboring 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 Figure 4.8 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. For instance, whenever multiple data qubits are available at each (or some nodes), only a subset of `CNOT`s must be mapped into remote operations. Hence, the compiler should be able to optimize choices such as which sub-circuit should be mapped to which node or which `CNOT`

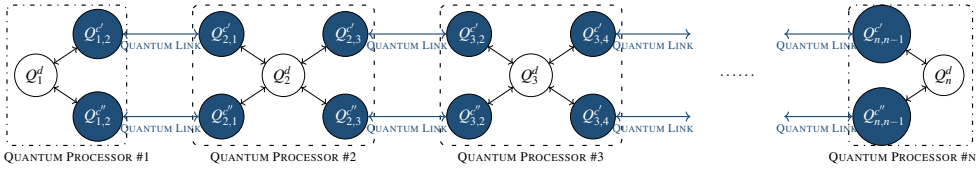
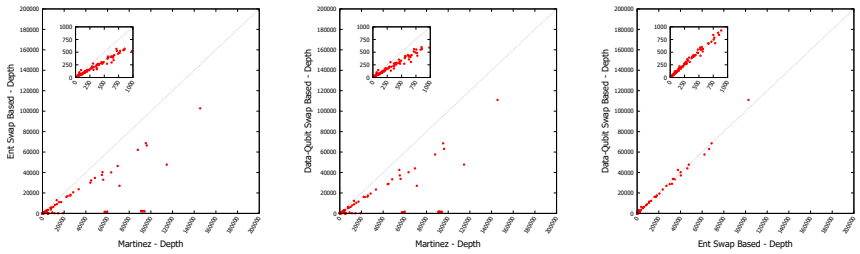


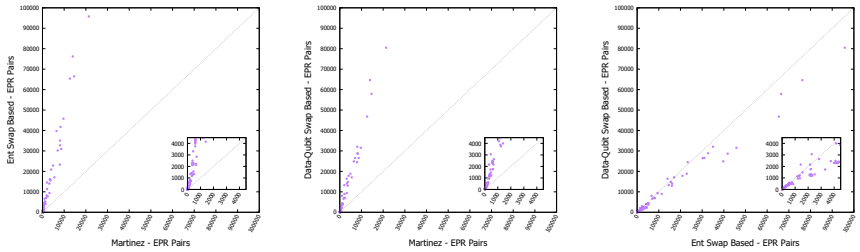
Figure 4.12: Improving over the worst-case scenario topology. Only one data qubit is available at each quantum processor, but neighboring quantum processors are connected with two quantum links, realized by 2 EPR pairs.

should be performed via communication qubits. Clearly, the optimal strategies – as well as the metrics to measure the optimality of a strategy – represent interesting open problems.



(a) **(b)** **(c)**

Figure 4.13: Comparing the *entanglement swapping based* and *data-qubit swapping based* strategies of our compiler with Andrés-Martínez's compiler [86], in terms of circuits' depth. Our compiler distributed circuits on the topology illustrated in Figure 4.12, with two links between neighboring QPUs, while Andrés-Martínez's one exploits a more favorable topology, i.e., the hypercube illustrated in Figure 4.7b. Both topologies are characterized by one data qubit per QPU.



(a) **(b)** **(c)**

Figure 4.14: Comparing the *entanglement swapping based* and *data-qubit swapping based* strategies of our compiler with Andrés-Martínez's compiler [86] over consumed EPR pairs. Our compiler distributed circuits on the topology illustrated in Figure 4.12, with two links between neighboring QPUs, while Andrés-Martínez's one exploits a more favorable topology, i.e., the hypercube illustrated in Figure 4.7b. Both topologies are characterized by one data qubit per QPU.

circuit name	# qubits n	# CNOT layers	Entanglement Swap			Data-Qubit Swap		
			# CNOT layers $\times \frac{q}{2} d_{es}$	compiled # layers	compiled/theoretical	# CNOT layers $\times (\frac{q}{4} d_{qs} + d'_{qs})$	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

Table 4.3: Validation of the theoretical upper bounds derived in Section 2.3.2 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.

4.4 Combining Local Routing And Remote Gate Scheduling

We tested the strategy proposed in Sec. 2.4, to combine local routing and remote gate scheduling, on two classes of quantum circuits, namely, VQE and QFT circuits. The aforementioned circuits have been compiled for the DQC architecture illustrated in Fig. 4.15, comprised of 5 QPUs, each characterized as in Fig. 4.16, with channel capacity set to 1, i.e., one communication qubit per connection. For each quantum circuit, we tested remote gate scheduling with only `TeleGate` and with `TeleGate` and `TeleData` operations.

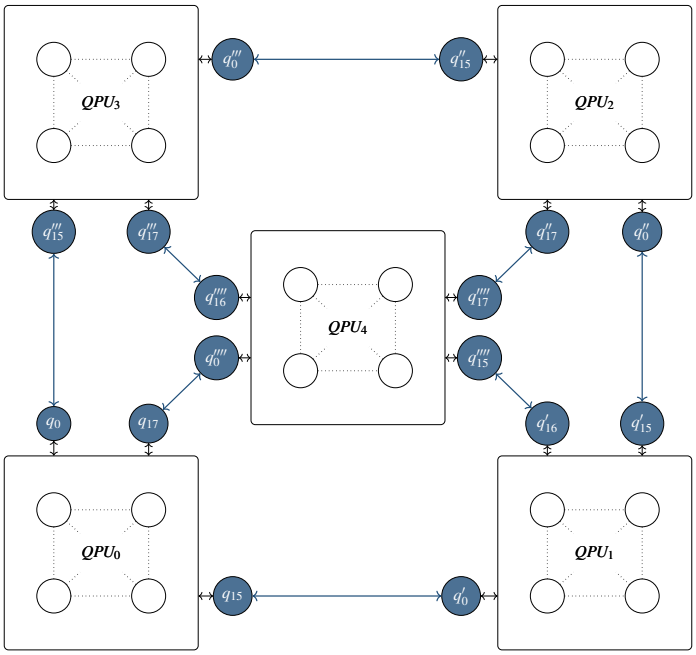


Figure 4.15: DQC architecture comprising 4 QPUs as shown in Fig. 4.16. Each QPU is connected to the others and each QPU supports 1 *communication qubit* per connection.

Fig. 4.17 and Fig. 4.18 show the depth of VQE and QFT circuits. It can be seen that exploiting `TeleData` operations alongside `Telegate`s tends to be more ben-

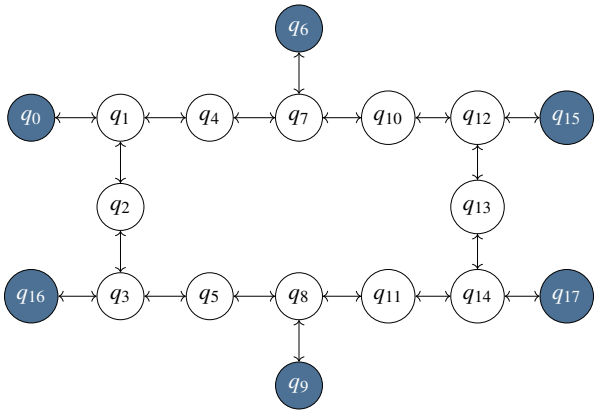


Figure 4.16: QPU configuration with 12 *data qubits* and 6 *communication qubits*, inspired by IBM's heavy hexagon devices [71].

eficial, in terms of total depth, for QFT circuits rather than the VQE ones.

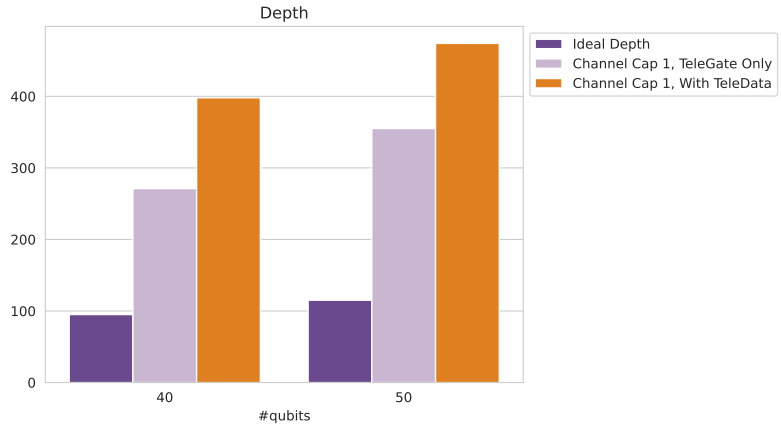


Figure 4.17: Depth of VQE circuits compiled for the DQC architecture of Fig. 4.15. The number of qubits varies from 40 to 50, while the channel capacity is set to 1.

Accordingly, as shown in Fig. 4.19 and Fig. 4.20, the use of both `TeleGate` and `TeleData` operations seems to be slightly detrimental for VQE circuits, at least for

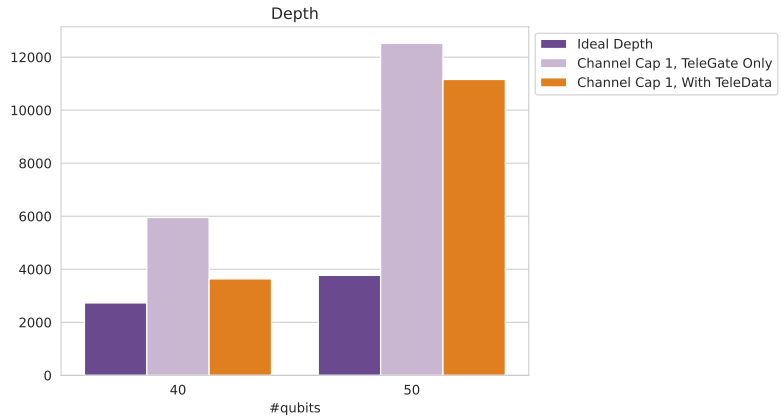


Figure 4.18: Depth of QFT circuits compiled for the DQC architecture of Fig. 4.15. The number of qubits varies from 40 to 50, while the channel capacity is set to 1.

---

these circuits instances on a small network, in terms of number of consumed EPR pairs and layers dedicated to remote operations.

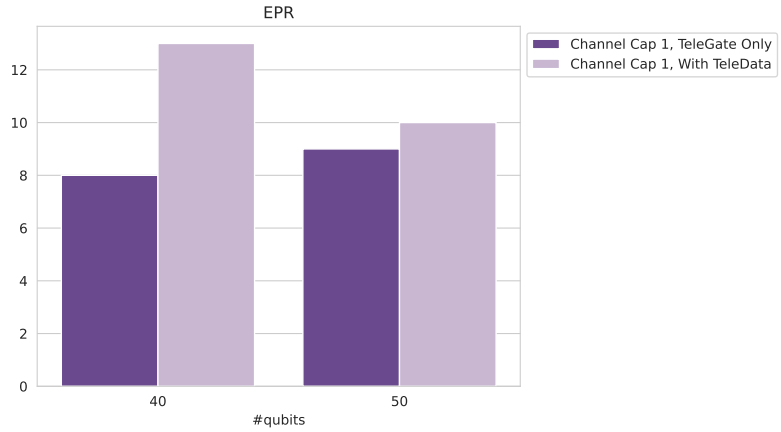


Figure 4.19: EPR pairs consumed to compile VQE circuits over the DQC architecture of Fig. 4.15. The number of qubits varies from 40 to 50, while the channel capacity is set to 1.

Conversely, QFT circuits can benefit from the use of `TeleData` both in terms

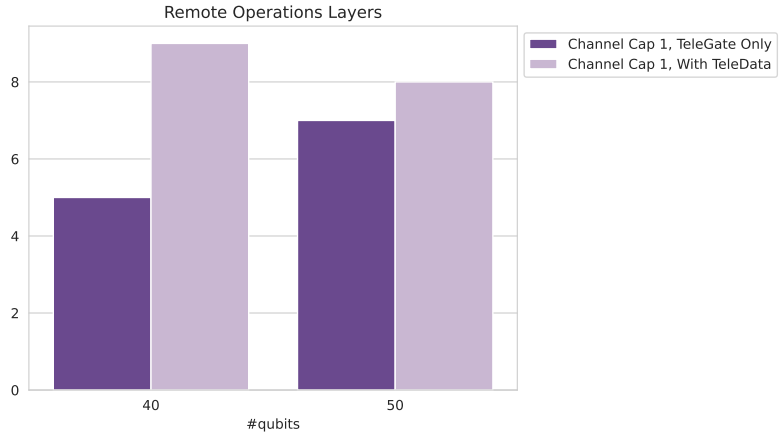


Figure 4.20: Number of layers dedicated to remote operations in VQE circuits compiled for the DQC architecture of Fig. 4.15. The number of qubits varies from 40 to 50, while the channel capacity is set to 1.

---

of consumed EPR pairs and layers dedicated to remote operations. As depicted in Fig. 4.21 and Fig. 4.22, there can be a reduction of up to 50% when using `TeleData` operations.

We also compiled the same quantum circuits on the DQC architecture presented in Sec. 2.1 (Fig. 2.2 and Fig. 2.3) while varying the channel capacity of each quantum link. Both Fig. 4.23 and Fig. 4.24 show a slight increase in total circuit depth when the channel capacity changes from 2 to 4. At first glance, this may seem counter intuitive and it is probably an overhead caused by the local routing, which tries to use all communication qubits available, regardless of their distance from data qubits in the local coupling map.

Regarding VQE circuits, it can be seen from Fig. 4.25 and Fig. 4.26 that, given the number of qubits, there is almost no difference between the various configurations of channel capacity and the use of `TeleData` operations.

Different considerations can instead be made with regard to the QFT circuits. Specifically, Fig. 4.27 clearly shows that, given the number of qubits, choosing to use both `TeleGate` and `TeleData` operations greatly reduces the number of con-

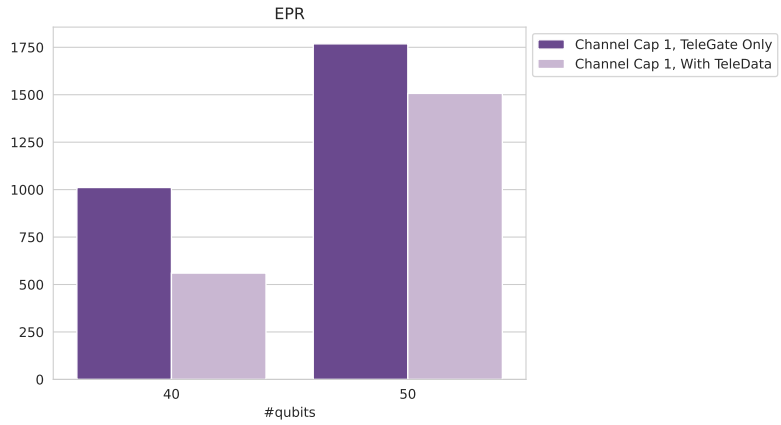


Figure 4.21: EPR pairs consumed to compile QFT circuits over the DQC architecture of Fig. 4.15. The number of qubits varies from 40 to 50, while the channel capacity is set to 1.

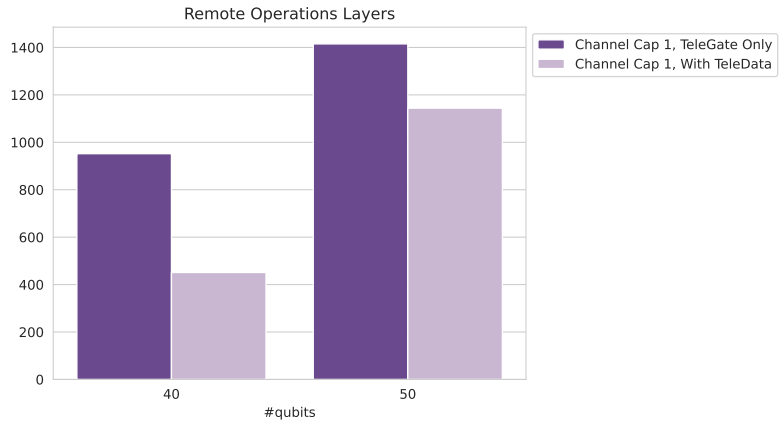


Figure 4.22: Number of layers dedicated to remote operations in QFT circuits compiled for the DQC architecture of Fig. 4.15. The number of qubits varies from 40 to 50, while the channel capacity is set to 1.

sumed EPR pairs. Of course, the number of consumed EPR pairs does not change when varying the channel capacity, as the number of remote operations is the same,

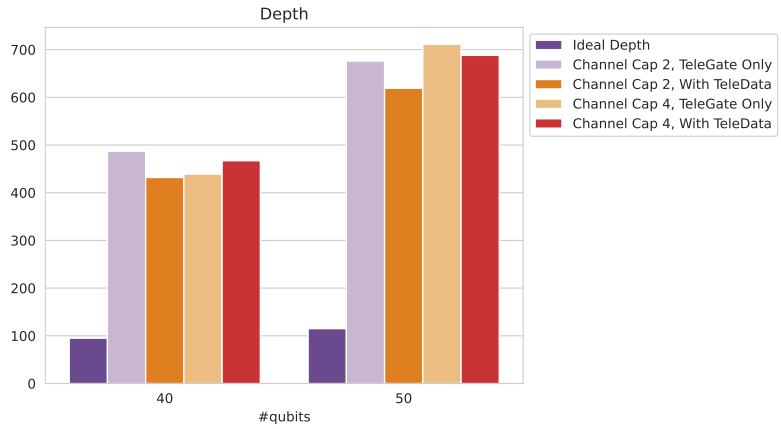


Figure 4.23: Depth of VQE circuits compiled for the DQC architecture of Fig. 2.3. The number of qubits varies from 40 to 50, while the channel capacity varies from 2 to 4.

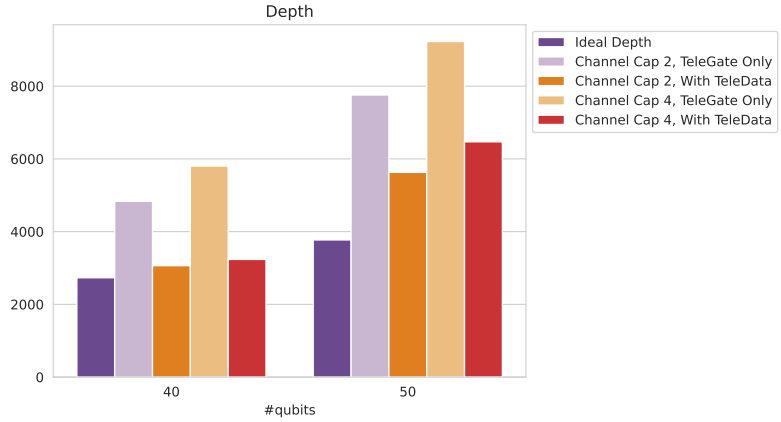


Figure 4.24: Depth of QFT circuits compiled for the DQC architecture of Fig. 2.3. The number of qubits varies from 40 to 50, while the channel capacity varies from 2 to 4.

regardless of channel capacity. Interestingly, there is also no difference in the number of layers dedicated to remote operations with respect to the channel capacity, as depicted in Fig. 4.28. We suppose that, due to the poor connectivity between data qubits

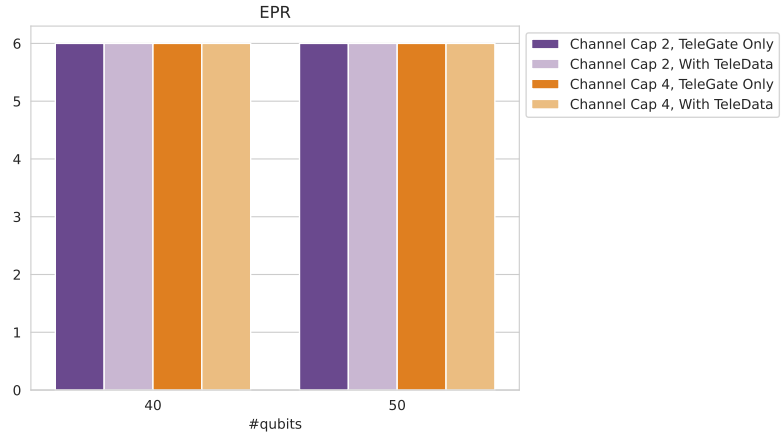


Figure 4.25: EPR pairs consumed to compile VQE circuits over the DQC architecture of Fig. 2.3. The number of qubits varies from 40 to 50, while the channel capacity varies from 2 to 4.

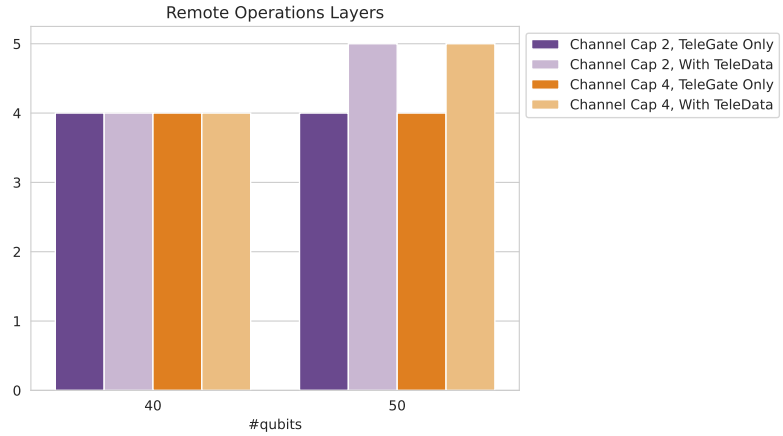


Figure 4.26: Number of layers dedicated to remote operations in VQE circuits compiled for the DQC architecture of Fig. 2.3. The number of qubits varies from 40 to 50, while the channel capacity varies from 2 to 4.

and communication qubits on each QPU, local routing operations create an upstream bottleneck with deleterious effects despite the increase in channel capacity. Further

investigations in this regard will be necessary.

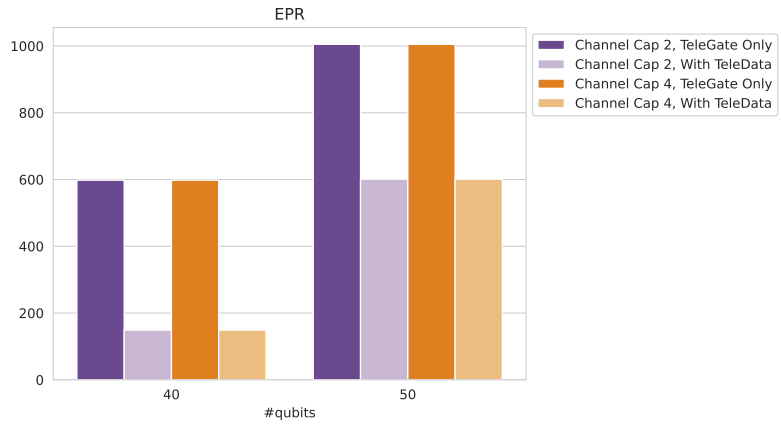


Figure 4.27: EPR pairs consumed to compile QFT circuits over the DQC architecture of Fig. 2.3. The number of qubits varies from 40 to 50, while the channel capacity varies from 2 to 4.

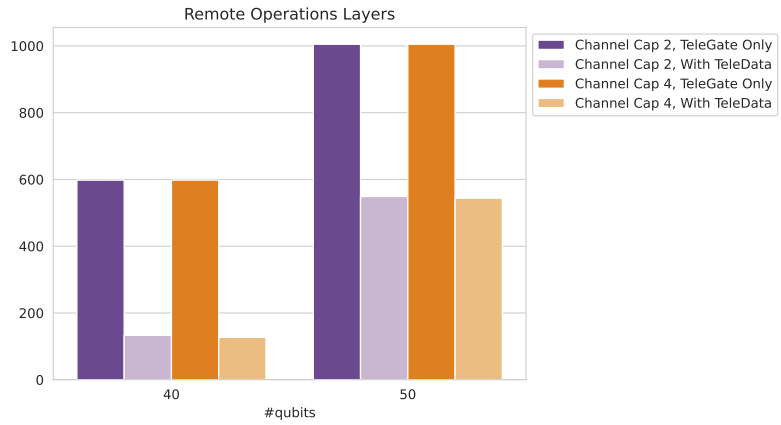


Figure 4.28: Number of layers dedicated to remote operations in QFT circuits compiled for the DQC architecture of Fig. 2.3. The number of qubits varies from 40 to 50, while the channel capacity varies from 2 to 4.

Conclusions

In this Thesis, we presented compilation strategies for local and distributed quantum computing, with the final goal of designing a general-purpose quantum compilation framework for distributed quantum computing.

In Sec. 2.2.1, we proposed novel deterministic algorithms for compiling recurrent quantum circuit patterns in polynomial time. Starting from this set of algorithms, we implemented PADQC, which has two main features. First, it identifies CNOT cascades and exploits useful circuit identities to transform them into CNOT nearest-neighbor sequences. Second, it finds an initial mapping that can comply with circuits characterized by recurrent CNOT nearest-neighbor sequences. Finally, we integrated PADQC with Qiskit's SABRE swapping strategy and compilation routine. We illustrated the results of the experimental evaluation of our integrated solution using different quantum programs and assuming IBM Quantum hardware. Among others, we compiled quantum circuits that are used to compute the ground state properties of molecular systems using the VQE method together with the RyRz heuristic wavefunction Ansätz. PADQC+Qiskit(SABRE), in general, produces output programs that are on par with those produced by state-of-art tools, in terms of CNOT count and CNOT depth. In particular, our solution produces unmatched results on RyRz circuits.

Then, in Sec. 2.3, we presented a computationally efficient noise-adaptive quantum compilation strategy. The contributed strategy assumes heavy-hexagon topologies for quantum devices, which is particularly crucial for the placement pass. Moreover, the assumption simplifies the derivation of the computational complexity upper

bounds. Nevertheless, the proposed routing pass is general enough to be effective independently of the coupling map of the target quantum device. Our compilation strategy is particularly effective for circuits characterized by great depth and/or randomness.

We also analytically derived an upper bound of the overhead induced by quantum compilation for distributed quantum computing, as detailed in Sec. 2.3.2. The derived bound accounts for the overhead induced by the underlying computing architecture as well as the additional overhead induced by the sub-optimal quantum compiler. To this aim, we designed a quantum compiler with three key features: i) general-purpose, namely, requiring no particular assumptions on the quantum circuits to be compiled, ii) efficient, namely, exhibiting a polynomial-time computational complexity so that it can successfully compile medium-to-large circuits of practical value, and iii) 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.

As the final goal of this Thesis is to propose a general-purpose quantum compilation framework, in Sec. 2.4 we presented a strategy to combine remote gate scheduling with local routing and then tested the resulting framework over a meaningful classes of quantum circuits, namely VQE and QFT ones. We also devised a strategy for remote scheduling that can exploit both `TeleGate` and `TeleData` operations and tested the impact of using either only `TeleGate`s or both. The evaluation results show that `TeleData` operations can have a positive impact on the number of consumed EPR pairs and, along with an augmented communication channel capacity, also on the number of layers dedicated to remote operations.

Regarding open problems, future work will focus on integrating noise-adaptive compilation strategies into the framework, both for local routing and remote gate scheduling. Consequently, we shall evaluate the impact of different strategies on the quality of computation results, which depend also on the selection of suitable metrics.

For instance, to discern if it is more beneficial to have a deeper compiled circuit that consumes less EPR pairs, or vice versa. This could be particularly useful for understanding the impact that remote operations, and any resulting local routing overhead, may have on the quality of the computation due to the effects of noise (such as decoherence). To produce such metrics we need to actually execute the compiled circuits, either by means of a quantum network simulator or, later on, on real hardware. In the first case, there are already available simulators with different levels of abstraction, depending on how realistic the simulations needs to be. For our purposes, it is crucial to realistically simulate the effects of different noise sources, as they can have a deep impact on the results of the computation and will be taken into account in future noise-adaptive compilation strategies. Regarding the execution on real quantum networks, small proof of concept quantum networks exist. The long-term ambition of the european project Quantum Internet Alliance (QIA) is to build a Quantum Internet that enables quantum communication applications between any two points on Earth. In this context, the proposed framework could be implemented and deployed as a service to support the execution of DQC jobs across the Quantum Internet.

Bibliography

[1] R. P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6-7):467–488, 1982.

[2] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[3] A. Chiesa, F. Tacchino, M. Grossi, P. Santini, I. Tavernelli, D. Gerace, and S. Carretta. Quantum hardware simulating four-dimensional inelastic neutron scattering. *arXiv e-prints*, page arXiv:1809.07974, 2018.

[4] V. Havlicek, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta. Supervised learning with quantum enhanced feature spaces. *arXiv e-prints*, page arXiv:1804.11326, 2018.

[5] F. Tacchino, C. Macchiavello, D. Gerace, and D. Bajoni. An artificial neuron implemented on an actual quantum processor. *arXiv e-prints*, page arXiv:1811.02266, 2018.

[6] C. Portmann and R. Renner. Cryptographic security of quantum key distribution. *arXiv e-prints*, page arXiv:1409.3525, 2014.

[7] A. K. Ekert. Quantum cryptography based on bell's theorem. *Phys. Rev. Lett.*, 67:661–663, 1991.

[8] J. Preskill. Quantum computing in the nisq era and beyond. *arXiv e-prints*, 2018.

[9] A. D. Córcoles, A. Kandala, A. Javadi-Abhari, D. T. McClure, A. W. Cross, K. Temme, P. D. Nation, M. Steffen, and J. M. Gambetta. Challenges and opportunities of near-term quantum computing systems. *Proceedings of the IEEE*, 108(8):1338–1352, 2020.

[10] A. Botea, A. Kishimoto, and R. Marinescu. On the Complexity of Quantum Circuit Compilation. In *The Eleventh International Symposium on Combinatorial Search (SOCS 2018)*, 2018.

[11] M. Soeken, G. Meuli, B. Schmitt, F. Mozafari, H. Riener, and G. De Micheli. Boolean satisfiability in quantum compilation. *Phil. Trans. Royal Soc. A*, 378(2164):1–16, 2019.

[12] A. Montanaro. Quantum algorithms: An overview. *npj Quantum Information*, 2(1):15023, January 2016.

[13] M. Caleffi, A. S. Cacciapuoti, and G. Bianchi. Quantum internet: From communication to distributed computing! In *Proceedings of the 5th ACM International Conference on Nanoscale Computing and Communication*, 2018. Invited Paper.

[14] M. Caleffi, D. Chandra, D. Cuomo, S. Hasaanpour, and A. S. Cacciapuoti. The Rise of the Quantum Internet. *IEEE Computer*, 2020.

[15] S. Pirandola and S. L. Braunstein. Physics: Unite to Build a Quantum Internet. *Nature*, 532(7598):169–171, Apr. 2016.

[16] E. Gibney. Chinese Satellite is One Giant Step for the Quantum Internet. *Nature*, 535(7613):478–479, July 2016.

[17] W Dür, R Lamprecht, and S Heusler. Towards a Quantum Internet. *European Journal of Physics*, 38(4):043001, May 2017.

[18] C. Simon. Towards a Global Quantum Network. *Nature Photonics*, 11(11):678–680, 2017.

[19] S. Wehner, D. Elkouss, and R. Hanson. Quantum internet: A vision for the road ahead. *Science*, 362(6412), 2018.

[20] M. Zomorodi-Moghadam, M. Houshmand, and M. Houshmand. Optimizing teleportation cost in distributed quantum circuits. *Int J Theor Phys*, 57:848–861, 2018.

[21] L. Gyongyosi and S. Imre. Entanglement concentration service for the quantum internet. *Quantum Information Processing*, 19(8):221, 2020.

[22] L. Gyongyosi and S. Imre. Routing space exploration for scalable routing in the quantum internet. *Scientific Reports*, 10(1):11874, 2020.

[23] D. Dieks. Communication by epr devices. *Physics Letters A*, 92(6):271–272, 1982.

[24] W. Wootters and W. Zurek. A single quantum cannot be cloned. *Nature*, 299:802–803, 1982.

[25] S. Wiesner. Conjugate coding. *ACM SIGACT News*, 15(1):78–88, 1983.

[26] B. Schumacher. Quantum coding. *Physical Review A*, 51(4):2738–2747, 1995.

[27] D. Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. *Proceedings of the Royal Society of London A*, 400(97):91–117, 1985.

[28] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2011.

[29] R. Jozsa and N. Linden. On the role of entanglement in quantum-computational speed-up. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 459(2036):2011–2032, 2003.

[30] J. S. Bell. On the Einstein Podolsky Rosen paradox. *Physics Physique Fizika*, 1:195–200, Nov 1964.

[31] A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? *Physical review*, 47(10):777, 1935.

[32] D. Gottesman. Theory of fault-tolerant quantum computation. *Phys. Rev. A*, 57:127–137, Jan 1998.

[33] J. Abhijith, A. Adedoyin, J. Ambrosiano, et al. Quantum algorithm implementations for beginners. *ACM Transactions on Quantum Computing*, 3(4), jul 2022.

[34] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, et al. Characterizing quantum supremacy in near-term devices. *Nature Physics*, 14(6):595–600, 2018.

[35] S. Koudia, A. S. Cacciapuoti, K. Simonov, and M. Caleffi. How deep the theory of quantum communications goes: Superadditivity, superactivation and causal activation. *IEEE Communications Surveys & Tutorials*, 2022. In press.

[36] A. S. Cacciapuoti, M. Caleffi, R. Van Meter, and L. Hanzo. When entanglement meets classical communications: Quantum teleportation for the quantum internet. *IEEE Transactions on Communications*, 68(6):3808–3833, 2020. invited paper.

[37] E. Rieffel and W. Polak. An introduction to quantum computing for non-physicists. *ACM Computing Survey*, 32(3):300–335, sep 2000.

[38] E. Rieffel and W. Polak. *Quantum Computing: A Gentle Introduction*. The MIT Press, 2011.

[39] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, 1996.

[40] A. W. Harrow, A. Hassidim, and S. Lloyd. Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.*, 103:150502, Oct 2009.

[41] M. Cerezo, A. Arrasmith, R. Babbush, et al. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, September 2021.

[42] B. M. Terhal. Quantum error correction for quantum memories. *Rev. Mod. Phys.*, 87:307–346, Apr 2015.

[43] J. I. Cirac and P. Zoller. Quantum computations with cold trapped ions. *Phys. Rev. Lett.*, 74:4091–4094, 1995.

[44] C. Adami and N. J. Cerf. Quantum computation with linear optics. In C. P. Williams, editor, *Quantum Computing and Quantum Communications*, pages 391–401", Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[45] G. K. Brennen, C. M. Caves, P. S. Jessen, and I. H. Deutsch. Quantum logic gates in optical lattices. *Phys. Rev. Lett.*, 82:1060–1063, 1999.

[46] D. Loss and D. P. DiVincenzo. Quantum computation with quantum dots. *Phys. Rev. A*, 57:120–126, 1998.

[47] A. Imamoglu, D. D. Awschalom, G. Burkard, D. P. DiVincenzo, D. Loss, M. Sherwin, and A. Small. Quantum information processing using quantum dot spins and cavity qed. *Phys. Rev. Lett.*, 83:4204–4207, 1999.

[48] L. Fedichkin, M. Yanchenko, and K. Valiev. Novel coherent quantum bit using spatial quantization levels in semiconductor quantum dot. *Quantum Computers and Computing*, 1(1):58–76, 2000.

[49] N. Ohlsson, R. Mohan, and S. Kröll. Quantum computer hardware based on rare-earth-ion-doped inorganic crystals. *Optics Communications*, 201:71–77, 2002.

[50] J. J. Longdell, M. J. Sellars, and N. B. Manson. Demonstration of conditional quantum phase shift between ions in a solid. *Phys. Rev. Lett.*, 93:130503, 2004.

[51] D. Stepanenko, M. Trif, and D. Loss. Quantum computing with molecular magnets. *Inorganica Chimica Acta*, 361(14):3740 – 3745, 2008.

[52] W. M. Kaminsky, S. Lloyd, and T. P. Orlando. Scalable superconducting architecture for adiabatic quantum computation. *arXiv e-prints*, 2004.

[53] J. Clarke and K. W. Frank. Superconducting quantum bits. *Nature*, 453(7198):1031–1042, 2008.

[54] Google Quantum AI. Suppressing quantum errors by scaling a surface code logical qubit. *Nature*, 614(7949):676–681, Feb 2023.

[55] R. Van Meter and S. J. Devitt. The path to scalable distributed quantum computing. *Computer*, 49(9):31–42, Sept 2016.

[56] D. Cuomo, M. Caleffi, and A. S. Cacciapuoti. Towards a distributed quantum computing ecosystem. *IET Quantum Communication*, 1:3–8, July 2020. Invited Paper.

[57] N. M. P. Neumann, R. van Houte, and T. Attema. Imperfect Distributed Quantum Phase Estimation. In *Computational Science – ICCS 2020*, Lecture Notes in Computer Science, pages 605–615. Springer International Publishing, 2020.

[58] A Y. Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191–1249, dec 1997.

[59] J. Eisert, K. Jacobs, P. Papadopoulos, and M. B. Plenio. Optimal local implementation of nonlocal quantum gates. *Phys. Rev. A*, 62:052317, Oct 2000.

[60] S. DiAdamo, M. Ghibaudi, and J. Cruise. Distributed Quantum Computing and Network Control for Accelerated VQE. *IEEE Transactions on Quantum Engineering*, 2:1–21, 2021.

[61] N. M. P. Neumann and R. S. Wezeman. Distributed quantum machine learning. In *Innovations for Community Services*, pages 281–293. Springer International Publishing, 2022.

[62] R. Parekh, A. Ricciardi, A. Darwish, and S. DiAdamo. Quantum algorithms and simulation for parallel and distributed quantum computing. In *2021 IEEE/ACM Second International Workshop on Quantum Computing Software (QCS)*, pages 9–19, Los Alamitos, CA, USA, nov 2021. IEEE Computer Society.

[63] J. Illiano, M. Caleffi, A. Manzalini, and A. S. Cacciapuoti. Quantum internet protocol stack: a comprehensive survey. *Computer Networks*, 213, 2022.

[64] R. Van Meter, K. Nemoto, W. J. Munro, and K. M. Itoh. Distributed arithmetic on a quantum multicomputer. In *33rd International Symposium on Computer Architecture (ISCA'06)*, pages 354–365, 2006.

[65] D. Ferrari, A. S. Cacciapuoti, M. Amoretti, and M. Caleffi. Compiler design for distributed quantum computing. *IEEE Transactions on Quantum Engineering*, 2:1–20, 2021.

[66] W. Kozlowski, S. Wehner, R. Van Meter, B. Rijsman, A. S. Cacciapuoti, M. Caleffi, and S. Nagayama. Architectural principles for a quantum internet. Internet-Draft draft-irtf-qirg-principles-10, Internet Engineering Task Force, 2022. Work in Progress.

[67] A. Kandala, K. Temme, A. D. Córcoles, A. Mezzacapo, J. M. Chow, and J. M. Gambetta. Error mitigation extends the computational reach of a noisy quantum processor. *Nature*, 567(7749):491–495, 2019.

[68] L. Gyongyosi and S. Imre. Circuit depth reduction for gate-model quantum computers. *Scientific Reports*, 10(1):11229, 2020.

[69] N. Raychev. Universal quantum operators. *International Journal of Scientific and Engineering Research*, 6(6):1369–1371, 2015.

[70] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta. Open quantum assembly language. *arXiv e-prints*, July 2017. arXiv:1707.03429.

[71] IBM. Ibm quantum systems. `https://quantum-computing.ibm.com/services/resources`, 2020.

[72] J. Majer, J. M. Chow, J. M. Gambetta, et al. Coupling superconducting qubits via a cavity bus. *Nature*, 449(7161):443–447, 2007.

[73] J. M. Chow, A. D. Córcoles, J. M. Gambetta, et al. Simple all-microwave entangling gate for fixed-frequency superconducting qubits. *Phys. Rev. Lett.*, 107:080502, Aug 2011.

[74] J. M. Chow, J. M. Gambetta, A. D. Córcoles, et al. Universal quantum gate set approaching fault-tolerant thresholds with superconducting qubits. *Phys. Rev. Lett.*, 109:060501, Aug 2012.

[75] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi. Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 1015–1029, New York, NY, USA, 2019. Association for Computing Machinery.

[76] S. Niu, A. Suau, G. Staffelbach, and A. Todri-Sanial. A Hardware-Aware Heuristic for the Qubit Mapping Problem in the NISQ Era. *IEEE Transactions on Quantum Engineering*, 1:1–14, 2020.

[77] S. Nishio, Y. Pan, T. Satoh, H. Amano, and R. Van Meter. Extracting Success from IBM's 20-Qubit Machines Using Error-Aware Compilation. *ACM Journal on Emerging Technologies in Computing Systems*, 16(3), May 2020.

[78] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan. t|ket⟩: a retargetable compiler for NISQ devices. *Quantum Science and Technology*, 6(1):014003, nov 2020.

[79] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, et al. Qiskit: An open-source framework for quantum computing, 2021.

[80] E. Munoz-Coreas and H. Thapliyal. Quantum circuit design of a T-count optimized integer multiplier. *IEEE Transactions on Computers*, 68(5):729–739, 2019.

[81] D. Mills, S. Sivarajah, T. L. Scholten, and R. Duncan. Application-Motivated, Holistic Benchmarking of a Full Quantum Computing Stack. *arXiv:2006.01273*, 2020.

[82] R. Blume-Kohout and T. C. Young. A volumetric framework for quantum computer benchmarks. *arXiv:1904.05546*, 2019.

[83] A. Zulehner, A. Paler, and R. Wille. An efficient methodology for mapping quantum circuits to the IBM QX architectures. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 38(7):1226–1236, 2019.

[84] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach 4th Edition*. Pearson, 2020.

[85] G. Li, Y. Ding, and Y. Xie. Tackling the qubit mapping problem for nisq-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 1001–1014, 2019.

[86] P. Andrés-Martínez and C. Heunen. Automated distribution of quantum circuits via hypergraph partitioning. *Phys. Rev. A*, 100:032308, Sep 2019.

[87] A. Yimsiriwattana and S. J. Jr. Lomonaco. Generalized GHZ states and distributed quantum computing. *Contemp. Math.*, 381, 2005.

[88] R. G. Sundaram, H. Gupta, and C. R. Ramakrishnan. Efficient Distribution of Quantum Circuits. In *35th International Symposium on Distributed Computing (DISC 2021)*, 2021.

[89] R. G. Sundaram, H. Gupta, and C. R. Ramakrishnan. Distribution of Quantum Circuits Over General Quantum Networks. In *2022 IEEE International*

Conference on Quantum Computing and Engineering (QCE), pages 415–425, 2022.

[90] O. Daei, K. Navi, and M. Zomorodi-Moghadam. Optimized quantum circuit partitioning. *Int J Theor Phys*, 59(12):3804–3820, December 2020.

[91] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

[92] Z. Davarzani, M. Zomorodi-Moghadam, M. Houshmand, and M. Nouri-baygi. A dynamic programming approach for distributing quantum circuits by bipartite graphs. *Quantum Information Processing*, 19, 2020.

[93] D. Dadkhah, M. Zomorodi, and S. E. Hosseini. A New Approach for Optimization of Distributed Quantum Circuits. *International Journal of Theoretical Physics*, 60(9):3271–3285, September 2021.

[94] E. Nikahd, N. Mohammadzadeh, M. Sedighi, and M. S. Zamani. Automated window-based partitioning of quantum circuits. *Phys. Scr.*, 96(3):035102, January 2021.

[95] D. Cuomo, M. Caleffi, K. Krsulich, F. Tramonto, G. Agliardi, E. Prati, and A. S. Cacciapuoti. Optimized compiler for distributed quantum computing, 2021.

[96] D. Ferrari, I. Tavernelli, and M. Amoretti. Deterministic algorithms for compiling quantum circuits with recurrent patterns. *Quantum Information Processing*, 20(6):213, Jun 2021.

[97] IBM. Quantum computation center opens. https://www.ibm.com/blogs/research/2019/09/quantum-computation-center/.

[98] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246, September 2017.

[99] A. Peruzzo, J. McClean, P. Shadbolt, M. H. Yung, X. Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(4213):1–7, 2014.

[100] P. Kl. Barkoutsos, J. F. Gonthier, I. Sokolov, N. Moll, G. Salis, A. Fuhrer, M. Ganzhorn, D. J. Egger, M. Troyer, A. Mezzacapo, S. Filipp, and I. Tavernelli. Quantum algorithms for electronic structure calculations: Particle-hole Hamiltonian and optimized wave-function expansions. *Phys. Rev. A*, 98:022322, Aug 2018.

[101] D. M. Greenberger, M. A. Horne, and A. Zeilinger. Going beyond bell's theorem. In M. Kafatos, editor, *Bell's Theorem, Quantum Theory, and Conceptions of the Universe*, pages 69–72. Kluwer Academic Publishers, 1989.

[102] S. Deffner. Demonstration of entanglement assisted invariance on IBM's quantum experience. *Heliyon*, 3(11), 2017.

[103] D. Ferrari and M. Amoretti. Efficient and effective quantum compiling for entanglement-based machine learning on ibm q devices. *International Journal of Quantum Information*, 16(08), 2018.

[104] Robert R. Tucci. QC Paulinesia. arxiv:0407215, July 2004.

[105] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Books on Computer Science. Dover Publications, 1998.

[106] D. Ferrari and M. Amoretti. Noise-adaptive quantum compilation strategies evaluated with application-motivated benchmarks. In *Proceedings of the 19th ACM International Conference on Computing Frontiers*, CF '22, page 237–243, New York, NY, USA, 2022. Association for Computing Machinery.

[107] IBM. IBM Quantum. https://quantum-computing.ibm.com/, 2021.

[108] C. Chamberland, G. Zhu, T. J. Yoder, J. B. Hertzberg, and A. W. Cross. Topological and subsystem codes on low-degree graphs with flag qubits. *Phys. Rev. X*, 10:011022, Jan 2020.

[109] R. W. Floyd. Algorithm 97: Shortest Path. *Commun. ACM*, 5(6):345, June 1962.

[110] M. Caleffi. Optimal routing for quantum networks. *IEEE Access*, 5:22299–22312, 2017.

[111] D. E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, September, 2 edition, 1998.

[112] T. Ono, R. Okamoto, M. Tanida, H. F. Hofmann, and S. Takeuchi. Implementation of a quantum controlled-swap gate with photonic circuits. *Scientific Reports*, 7(1):45353, Mar 2017.

[113] N. Schuch and J. Siewert. Natural two-qubit gate for quantum computation using the XY interaction. *Phys. Rev. A*, 67:032301, Mar 2003.

[114] A. Dahlberg, B. Van Der Vecht, C. Delle Donne, M. Skrzypczyk, I. Te Raa, W. Kozlowski, and S. Wehner. Netqasm—a low-level instruction set architecture for hybrid quantum–classical programs in a quantum internet. *Quantum Science and Technology*, 7(3):035023, jun 2022.

[115] F. Arute, K. Arya, R. Babbush, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, Oct 2019.

[116] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. Revlib: An online resource for reversible functions and reversible circuits. In *38th International Symposium on Multiple Valued Logic (ismvl 2008)*, pages 220–225, 2008. RevLib is available at http://www.revlib.org.

[117] A. S. Green, P. F. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron. Quipper: A scalable quantum programming language. *SIGPLAN Not.*, 48(6):333–342, June 2013.

[118] A. Javadi-Abhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F. T. Chong, and M. Martonosi. Scaffcc: Scalable compilation and analysis of quantum programs. *Parallel Computing*, 45:2 – 17, 2015. Computing Frontiers 2014: Best Papers.

Acknowledgments

I would first like to express my gratitude to my supervisor Prof. Michele Amoretti for his patient guidance, encouragement and useful critiques to this work. To my friends Andrea, Clarissa, Clelia and Jacopo, thank you for listening and enduring me through this entire process. To my colleagues and now friends Andrea, Irene, Marina, Mattia and Stefano, who have put up with me in these three years. Last but not least, I am deeply grateful to my parents for their unfailing support and continuous encouragement throughout my life and studies.