araQne: a compiler for distributed quantum computing

The practical implementation of quantum computing to **solve real-world industrial problems** is one of the most highly anticipated milestones in the field. Applications span diverse areas, including vehicle routing, molecular design, portfolio optimization, and cryptography. Achieving this goal requires quantum computers with both high qubit capacity and low error rates. For instance, both breaking the RSA-2048 encryption algorithm and developing corrosion-resistant materials are estimated to require orders of magnitude of thousands of qubits. Additionally, a significant challenge lies in quantum data's inherent fragility, which necessitates error correction for achieving low error rates. Error correction involves bundling hundreds or thousands of physical qubits into a single, reliable logical qubit to reduce errors. This process dramatically inflates the number of physical qubits needed. For the examples mentioned, the number of physical qubits scales polynomially with the logical qubits, pushing the required physical qubit count into the millions. Thus, scaling is the key to unlock quantum computing's transformative potential in industrial applications. However, as number of qubits within a single *Quantum Processing Unit* (QPU) grows, issues like qubit decoherence, dissipation, and crosstalk intensify and compromise the quantum behavior of the system.

Distributed Quantum Computing (DQC) has emerged as a promising approach to providing an alternative direction towards scalability of current quantum systems. The idea behind it is the distribution of algorithms over a network of interconnected QPUs. In this context, the problem of **automating and optimizing this distribution** arises. We at Welinq are at the forefront of current DQC research with pioneering research and development on hardware and algorithms and are glad to present araQne: a compiler for distributed quantum computing, currently under development by our algorithm group. This presentation provides a first glimpse of the **current capabilities** of the compiler and how it enhances the applicability of quantum algorithms through efficient distribution strategies.

DQC in a Nutshell

DQC envisions quantum processors interconnected via quantum links and classical channels. A *quantum link* (QL) is needed to create entanglement between qubits belonging to different QPUs. Specifically, a single query to the quantum link allows QPUs to share a *Bell Pair* (BP):

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}.$$

In this framework, we distinguish between two types of qubits: **data qubits** and **communication qubits**. The former are exclusively dedicated to performing the quantum computations outlined by the algorithm. The latter, on the other hand, are dedicated to communication between QPUs, as they are reserved in storing BPs.

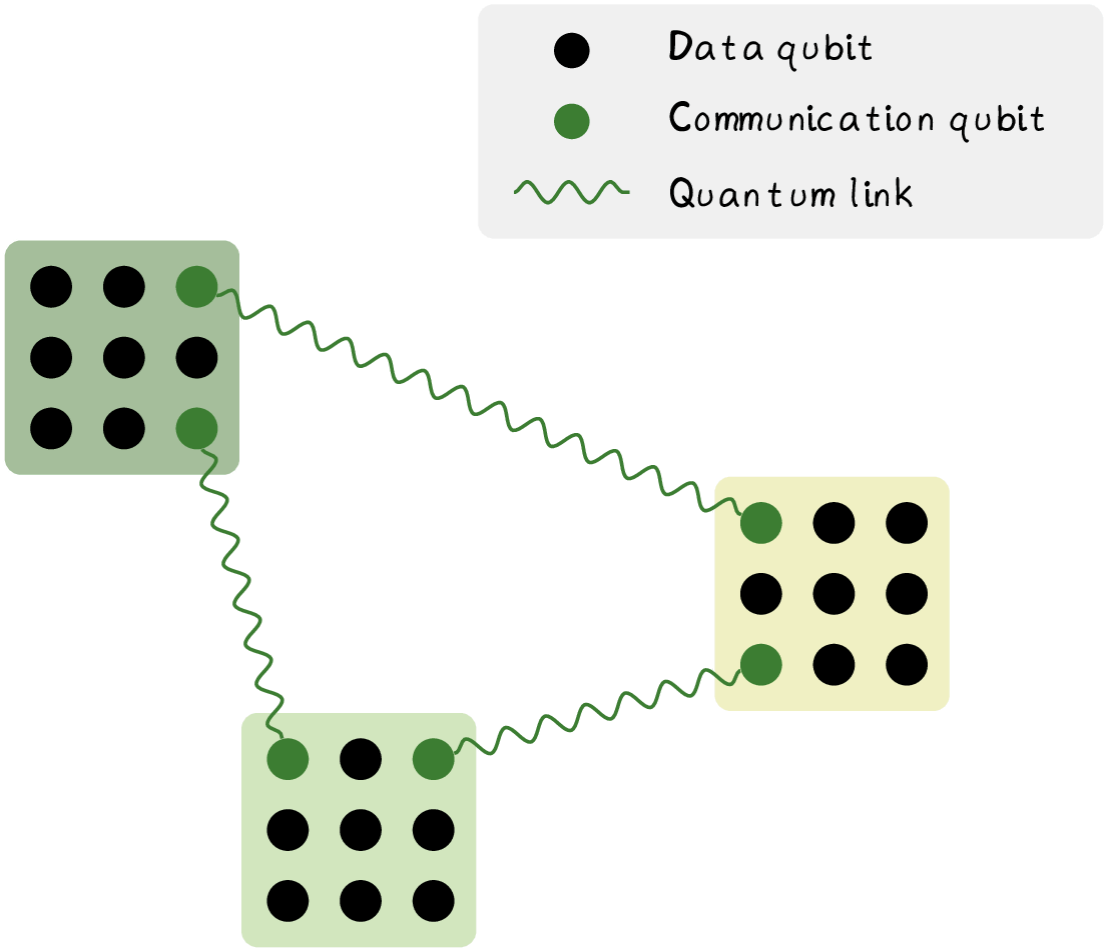


Figure: network of distributed and interconnected QPUs.

Implementation of non-local operations

These **interconnected QPUs should work as a single quantum processor** capable of executing algorithms that demand a number of qubits beyond the capacity of a single current device. In order to achieve this an algorithm should be divided into smaller ones, each of which is assigned to a different QPUs. When this is done certain multi-qubit operations may involve qubits located on distinct QPUs. We refer to these operations as non-local.

How is it possible to implement a non-local operation? It is sufficient that a Bell state is stored into a pair communication qubits, one per QPU. Once the quantum link is used, only *Local Operations and Classical Communication* (LOCC) are needed to carry out a non-local operation.

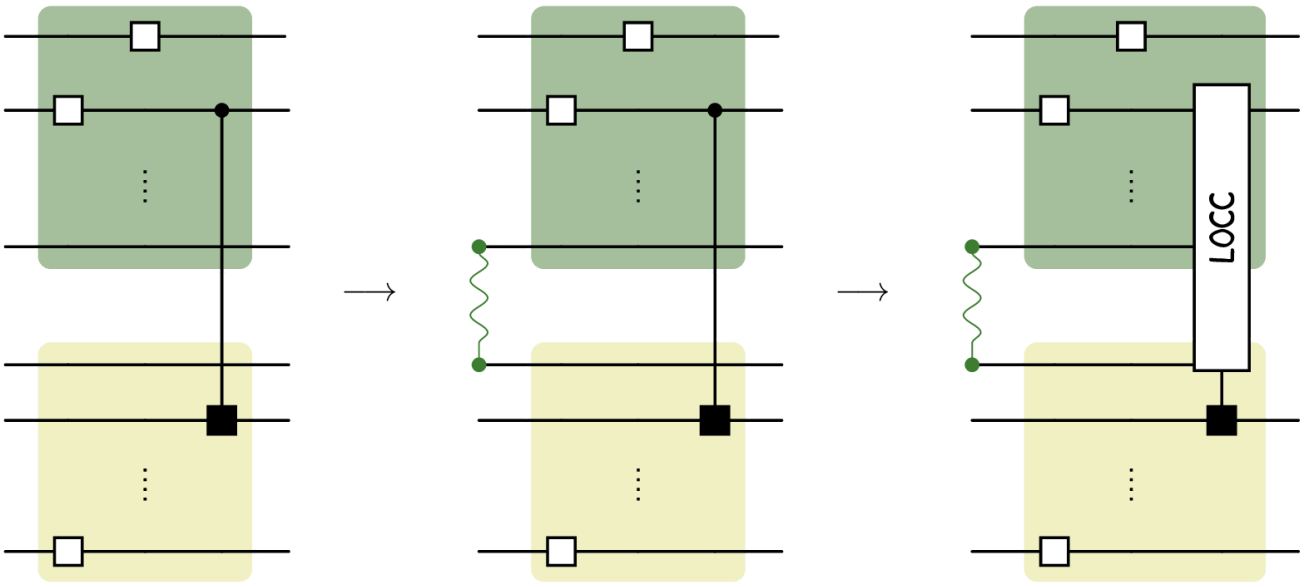


Figure: non-local gate implementation via quantum communication protocol.

Let us be more concrete and see one example. Suppose that we want to implement a CNOT on two qubits assigned to separate QPUs. One way to to it is to use the so-called *TeleGate* (TG) *protocol*, which uses two essential communication primitives called *Cat-Entangler* (CE) and *Cat-Disentangler* (CD).

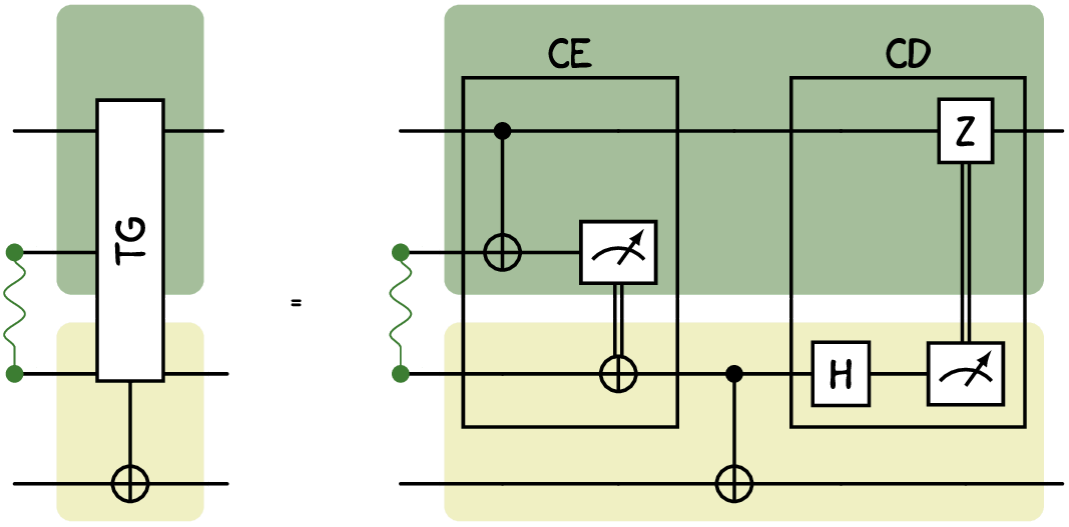


Figure: TG protocol for implementing a non-local CNOT.

We remark that, according to the TG protocol shown here, one BP is consumed when a single non-local operation is performed. Because the number of quantum links may be limited in real hardware and the depth of the algorithm is slightly increased after each TG, the **number of uses of quantum links is a key parameter** to optimize when distributing a quantum algorithm.

A Compiler for DQC

Given what we have seen up to now, a key question arises: what is an optimal way to partition a quantum circuit over many QPUs so as to **minimize the number of timed a Bell pair is consumed**?

araQne is designed with the goal of answering this question, seeking to provide optimal partitioning of a given circuit. That is, a partitioning that reduces as much as possible the entanglement resources required to implement the necessary nonlocal operations. At the current state, our compiler addresses **two core challenges**:

- qubit allocation: how to decide which qubit should be allocated to which QPU,
- job scheduling: how to apply non-local gates in a way that minimizes the number of inter-processor operations between different QPUs

These two challenges become increasingly hard to solve as the quantum algorithms scales up, hence the need for an automatic tool such as araQne.

DQC compiler workflow

araQne workflow begins by the input information, comprised by two parts, as shown in the figure below: a **quantum algorithm** represented as an abstract circuit and the **hardware specifications** such as number of QPUs in the network or available data qubits in each QPU. For simplicity in the following we will only consider a balanced scenario, where each QPU has the same amount of data qubits, and no restriction on the number of communication qubits or network topology.

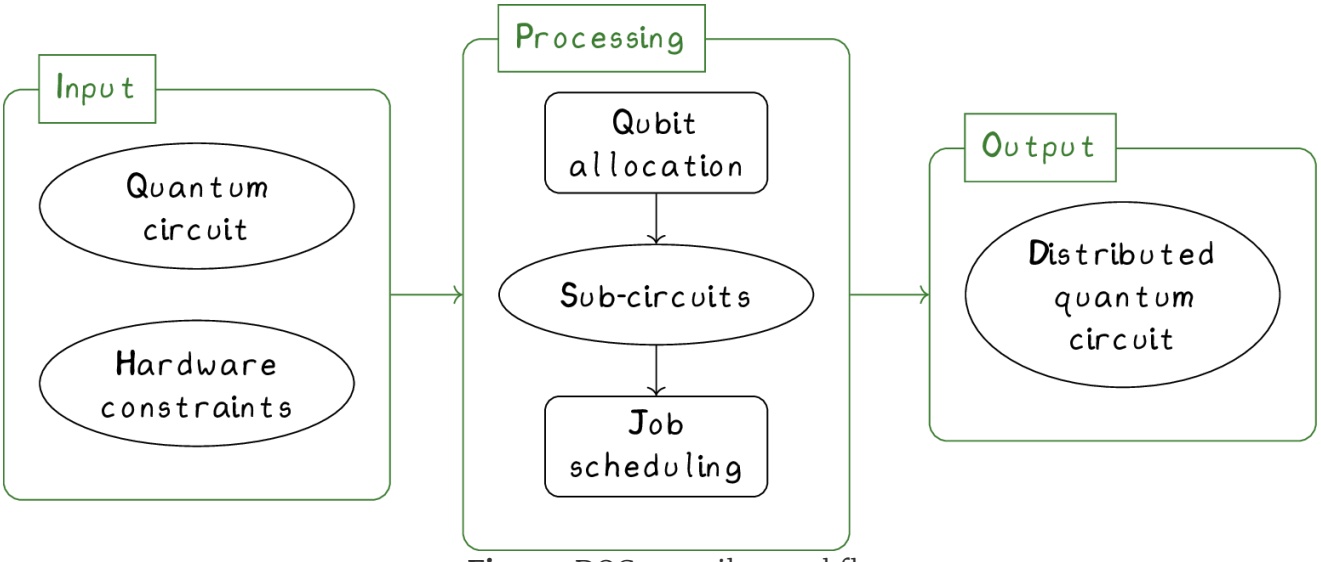


Figure: DQC compiler workflow.

araQne processing phase starts by **partitioning the monolithic circuit** to address the qubit allocation problem, ensuring that each QPU receives the same amount of qubits. The resulting non-local gates between sub-circuits are then analyzed in order to **minimize quantum communication**, meaning that entanglement resources

consumed by operations between qubits on different QPUs are reduced as much as possible, hence addressing the remote gate scheduling problem as well. In its final output, the compiler produces a **distributed version of the original algorithm**.

This work-flow enables a seamless translation from a monolithic quantum circuit to an optimized, distributed version that maximizes computing capabilities while minimizing communication overhead.

In the following we will have a look at how a DQC compiler could help us transform an undistributed monolithic quantum circuit into its distributed version.

Distributed quantum Fourier transform

The *Quantum Fourier Transform* (QFT) algorithm has proven to be central as a key building block for many quantum algorithms: first and foremost, it permits access to the estimation of the eigenvalues of unitary operators via the *quantum phase estimation* algorithm, as well as to Shor's algorithm for the factoring of large integers, while also to order finding and solution counting algorithms, once combined with the quantum search algorithm. Its successful implementation is therefore very important to a wide range of applications in quantum computing.

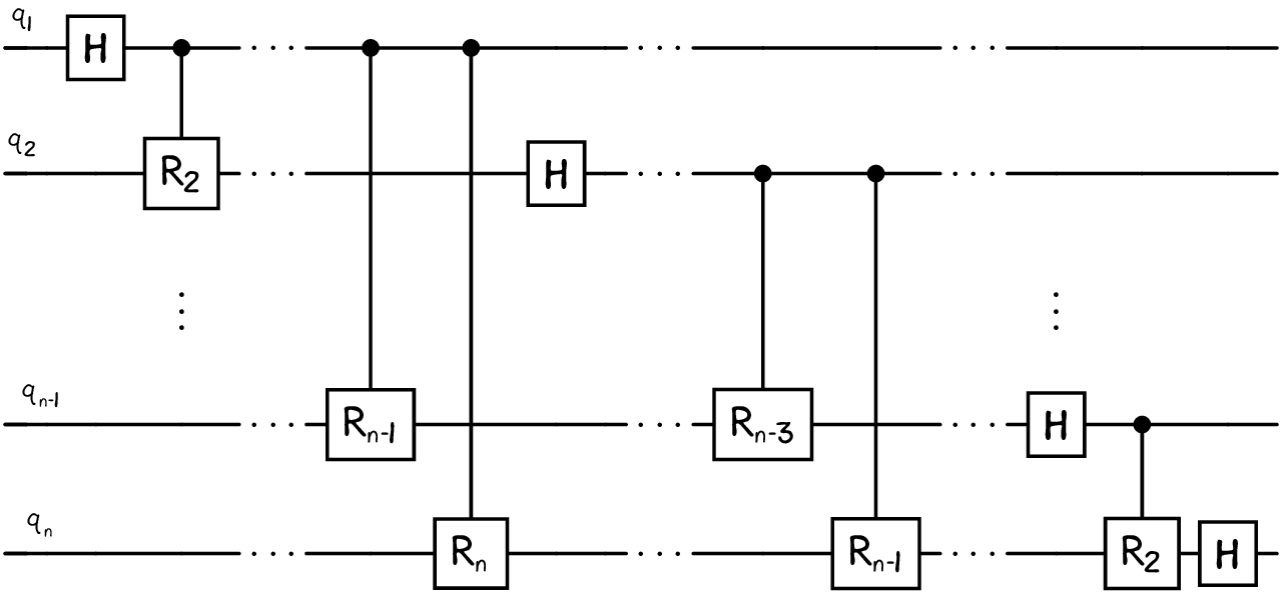


Figure: QFT circuit for *n* qubits.

We focus here on the **standard implementation** of QFT, as it is an example of circuit with practical use and for which the qubit connectivity is a complete graph, that is, the worst case scenario for the distribution of a monolithic quantum circut.

Non-local gates in QFT

QFT is an excellent candidate to test DQC due to its recursive and layered structure which includes a high number of two-qubit gates. Such controlled gates become increasingly challenging to manage from a distributed perspective, as the portion of those that have to be non-local increases rapidly as the number of qubits grows.

Indeed, consider the QFT algorithm for n logical qubits and suppose we want to distribute it over k QPUs, where k is a dividend of n, each of which can handle the same number of qubits. Since there are $(n/k)^2$ non-local gates between every pair of partitions and $k(k-1)/2$ pairs of possible partitions, then a balanced k-partition introduced a **number of non-local gates** equal to

$$\frac{k(k-1)}{2}\left(\frac{n}{k}\right)^2,$$

each of which needs to be implemented using a protocol.

As explained before, in general the number of Bell pairs matches that of non-local operations in the distributed circuit. The key to achieving this reduction is to optimally identify sequences of controlled gates sharing the same control qubit, with target qubits managed by different QPUs, as shown in the picture below, and apply TG to the whole sequence.

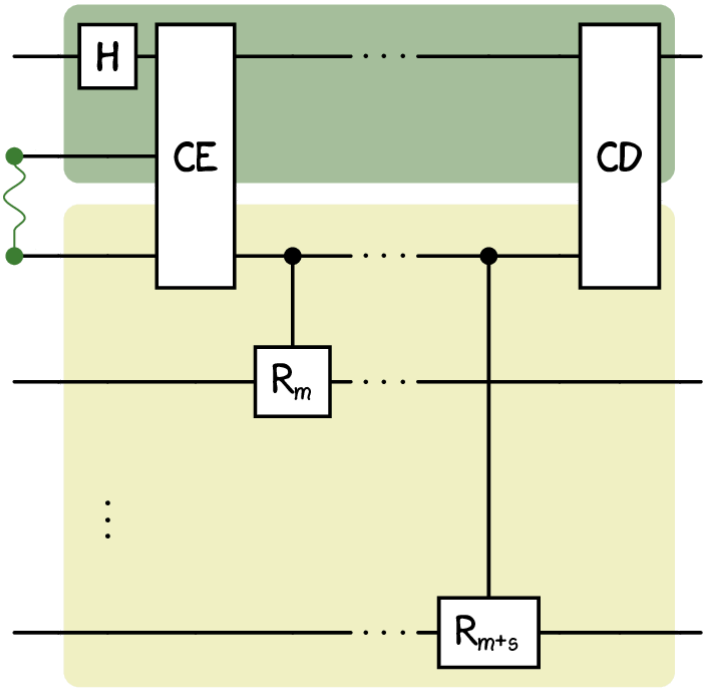
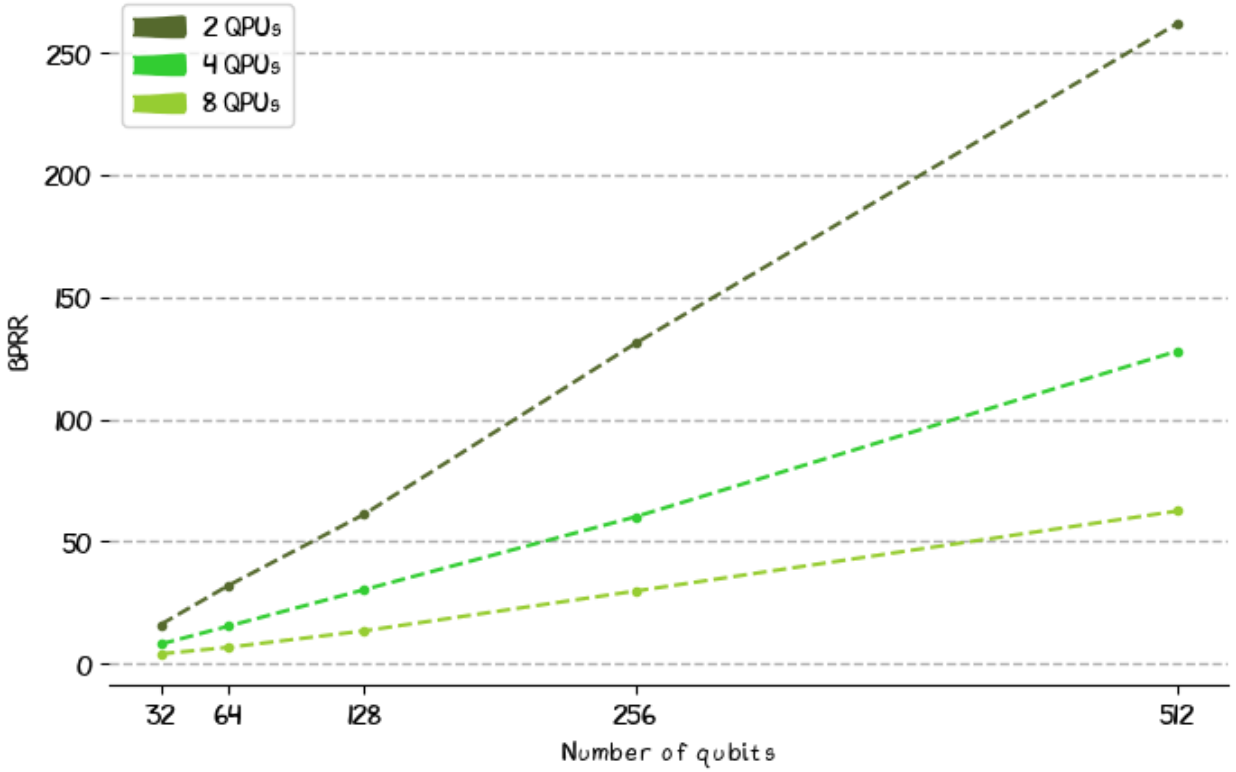


Figure: TG protocol covering more than one non-local gate.

With this in mind, let us turn our attention to the results achieved by araQne, which leverages such property of TG by using a **smart sequence identification strategy**. We quantify the difference between the number N_{NL} of non-local gates and the number N_{BP} of Bell pairs consumed for their implementation by the *Bell Pair Reduction Ratio* ($BPRR$), defined as follows:

$$BPRR = \frac{N_{NL}}{N_{BP}}.$$

It quantifies how many non-local gates can implemented with one single BP. Without the above strategy the BPRR is 1, signaling that each non-local operation is implemented using a BP. On the contrary, with the above strategy, as it can be seen in the plot below, indicates that a single BP can be used for the implementation of multiple non-local operations covered by the same TG.



Picture: *BPRR* per number of qubits for QFT. Colors identify different *k*-partitions, with *k* being the number of QPUs.

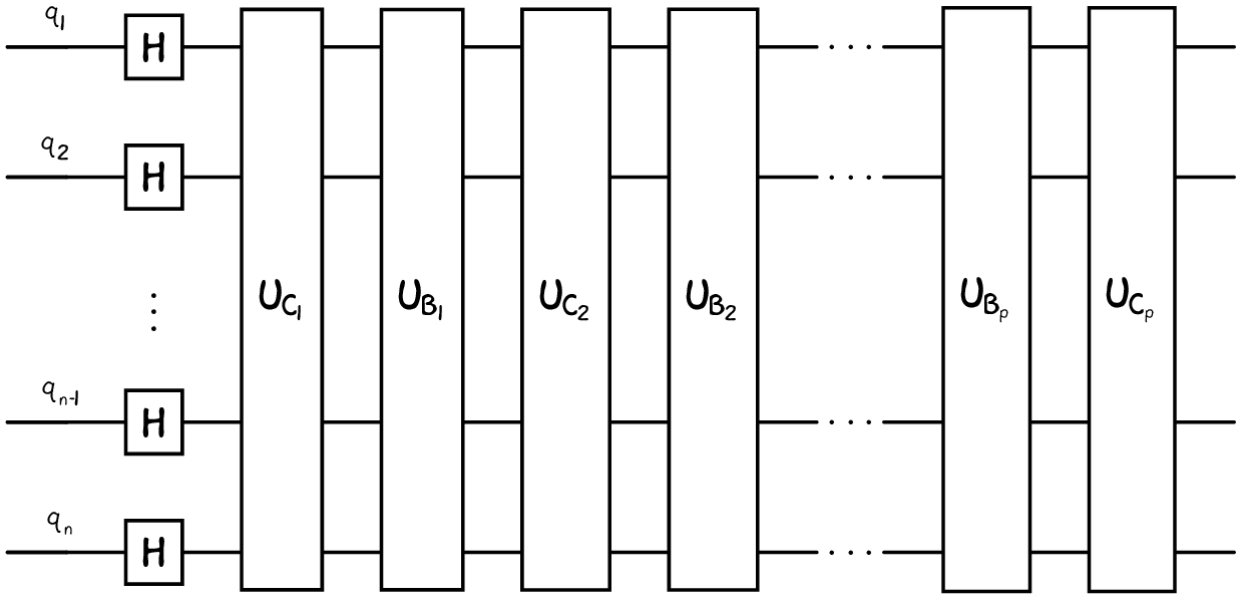
As it can be seen in the picture above, araQne reduces the number of Bell pairs required by a **factor that grows with the number of qubits** in the circuit. Finally, note that the distribution becomes more expensive, in terms of Bell pairs, as the number of QPUs in the network grows. Therefore, the BPRR decreases for a fixed circuit size and an increasing number of QPUs.

Distributed quantum approximate optimization algorithm

The *Quantum Approximate Optimization Algorithm* (QAOA) is a quantum algorithm designed for tackling optimization problems which is proven to be highly useful for implementation in current NISQ devices. However, large-scale quantum systems are required to represent complex interactions among variables which are usually present in real-world scenarios. A **distributed setup may allow for scalability** beyond the limitations of individual QPUs, enabling QAOA to tackle problem sizes and complexities that would be unattainable on a single QPU.

One relevant example is the *Max-Cut* (MC) problem. The MC problem has many applications, from classification problems in machine learning to optimization problems in finance, logistics, physics and circuit design. Given an undirected graph G, the Max-Cut problem is solved by identifying two partitions of G such that the number of cut edges is maximized.

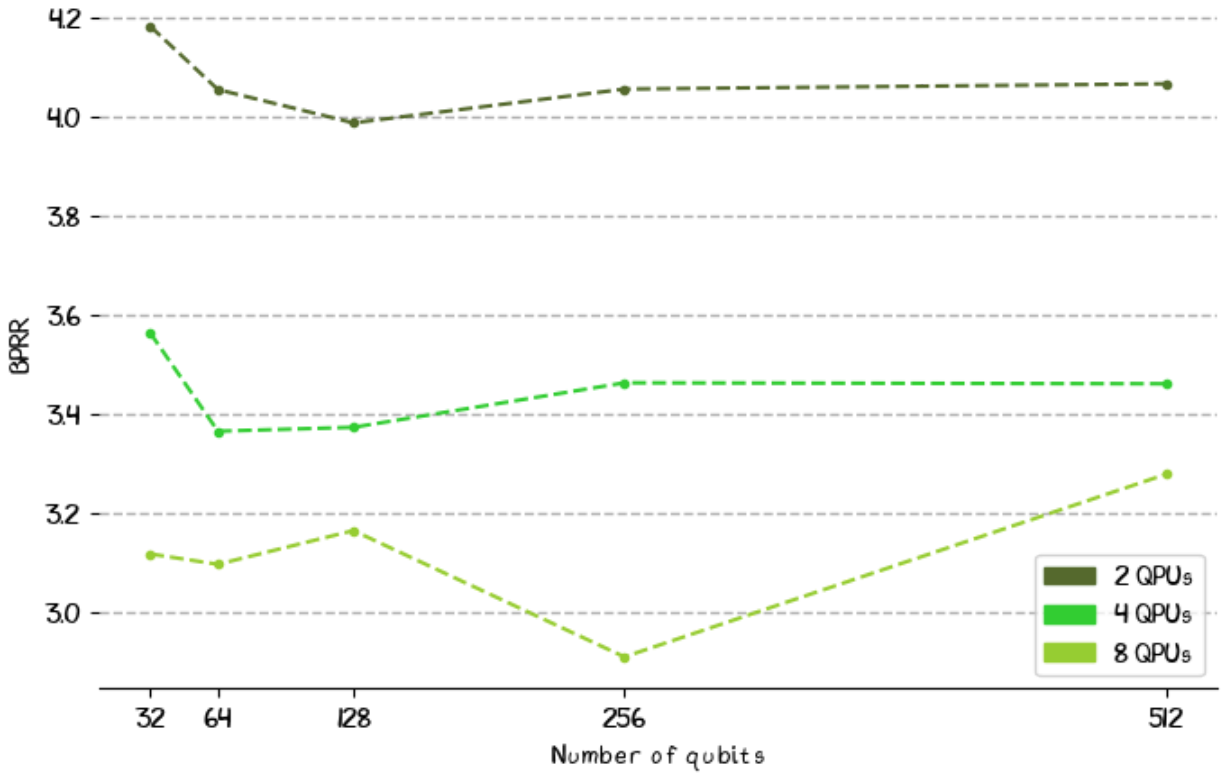
We can use QAOA to tackle this problem, employing unitary operators $U_{B_p}, U_{C_p}, \ldots, U_{B_1}, U_{C_1}$ which are Hamiltonian oracles, parametrized by $\gamma = (\gamma_1, \ldots, \gamma_p)$ and $\beta = (\beta_1, \ldots, \beta_p)$. At each layer l, U_{B_l} and U_{C_l} encode respectively the mixer Hamiltonian, which explores the solution space, and the cost Hamiltonian, which encodes the optimization problem.



Picture: QAOA Circuit composed of *p* layers.

As was done for QFT, it is possible to consider the required Bell pairs in the distribution and compare them with the total number of nonlocal gates for a single layer of the QAOA circuit. This is shown in the picture below, where we consider *Erdős–Rényi* (ER) random graph instances with an increasing number of vertexes and with probability 0.8 of edge inclusion. To measure the ratio of Bell pair saved by araQne, the $BPRR$ metric is again employed. In this case, the number of nonlocal

gates is calculated using a naïve qubit assignment technique based on simple weighted graphs.



Picture: *BPRR* per number of qubits for QAOA applied to ER graphs with probability 0.8. Colors identify different k-partitions, with k being the number of QPUs.

Here, the number of Bell pairs consumed in the distributed QAOA is **reduced down to one forth** thanks to araQne. The difference between the scaling of the BPRR in the QFT and QAOA cases stems from the fact that the structure of the former algorithm allows greater coverage of non-local operations by a single TG protocol.

Summary

We have introduced the **araQne compiler** and its use for DQC. For the algorithms presented here, QFT and QAOA, we witness a **significant optimization** of the use of entanglement resources, achieved by employing specific optimization techniques. Future versions of araQne will incorporate additional such techniques, along with the possibility of giving as input network topologies and more hardware-specific features.