# OPTIMIZING QUANTUM CIRCUITS FOR EFFICIENT EXECUTION ON SUPERCONDUCTING HARDWARE: A CASE STUDY OF THE CUCCARO ADDER

TECHNICAL REPORT

**Ricard-Santiago Raigada-García***
Department of IT, Multimedia and Telecommunications
Universitat Oberta de Catalunya
Rambla del Poblenou, 156, 08018, Barcelona, Spain.
rraigadag@uoc.edu

April 5, 2024

**ABSTRACT**

This technical report delves into the practical aspects of efficiently implementing quantum algorithms on superconducting quantum hardware. A theoretical and practical framework on the compilation of quantum programs is offered. In particular, three fundamental compilation steps are covered: qubit mapping, routing, and gate scheduling. Essential for translating high-level algorithms into real hardware implementations. The Cuccaro adder is used, a contribution to quantum arithmetic that allows efficient addition to be performed using a single ancilla qubit. This report shows the practical application of these compilation phases, with special emphasis on optimizing circuit performance for superconducting quantum hardware.

***Keywords*** Quantum Computing · Quantum Circuit Compilation · Qubit Mapping · Routing · Gate Scheduling · Superconducting Quantum Hardware · NISQ Era · Cuccaro Adder · Optimization Techniques

## 1 Introduction

Quantum computing is surpassing the current physical capabilities that limit classical computing, including supercomputers. In fact, currently on AWS you have access to QuEra—Aquila which has 256 qubits based on neutral atoms. Classical supercomputers can try to simulate at most 50 to 100 qubits Preskill [2012]. Therefore, one of the main benefits is high computing power (quantum supremacy). It has been announced by large companies in the sector that in no more than a decade they will have quantum computers with around 1000 qubits (governed by Moore's law).

However, it is only one of the many benefits. Among these, I can highlight the double storage capacity or computational space due to the superposition capacity of classical bits, and the benefit of being governed by the laws of quantum mechanics to model events that are given in nature itself. Particularly, in the field of quantum machine learning, one of the great advantages is adiabatic quantum computing, since it is capable of finding the global optimum of non-convex objective functions Wittek [2014]. All of this translates into the ability to model and predict natural events from your own real simulation. In fact, information theory suggests that the universe is a quantum computer and that the world can be encoded by qubits. This is explained by Seth Lloyd, professor at MIT, in the book Programming the universe.

### 1.1 Purpose and Scope of Work Performed

This work focuses on aspects related to the efficient and real implementation of quantum algorithms on quantum hardware. Therefore, it offers a theoretical and practical framework for the compilation of quantum programs. Compilation involves going from a high-level description of a quantum algorithm to a sequence of operations executable and interpretable by a quantum processing unit. Fundamentally, maximum efficiency, maximization of quantum resources, parallelization of quantum gates and maximization of quantum coherence are required.

*https://thedatascientist.digital/

The work is specifically focused on three fundamental compilation steps: mapping, routing, and quantum gate scheduling. These three steps are fundamental to translating a quantum algorithm to implementation on real hardware. Likewise, ensuring the efficient and effective implementation of the resulting circuits for maximum performance optimization.

## 2 Theoretical fundament

### 2.1 Qubit Mapping

Logical qubits must be mapped to physical qubits in a quantum processor with the goal of minimizing the number of additional operations required. The mapping of qubits faces different complexities that involve everything from decoherence due to applying an excess of gates, decoherence due to an incorrect physical distribution strategy to connectivity between qubits regarding quantum architecture. Effective mapping involves minimizing the computational cost associated with a quantum circuit.

To achieve a correct mapping of logical to physical qubits, hardware connectivity and program interaction are considered. To evaluate hardware connectivity, it is usually done using graphs that represent the physical connections between qubits in a processor.

To achieve this, the physical qubits are represented as nodes, and the edges represent the ability to perform two-qubit operations in parallel. In relation to the interaction of the program, the dependencies and interactions between logical qubits in a quantum circuit are evaluated. Each node represents the logical qubit, and the edges represent two-qubit operations.

### 2.2 Qubit Routing

Once qubit mapping is done, routing performs circuit tuning so that two-qubit operations only occur between adjacent physical qubits in the hardware. In the context of NISQ machines, commonly when you have a long-distance two-qubit gate, it is solved by moving one of the qubits closer to the other through a chain of gates SWAP Ding and Chong [2020].

### 2.3 Scheduling gate

Gate scheduling is the sequencing of quantum operations in time that aims to minimize the total execution time of a circuit. Likewise, allowing the parallelization of quantum operations. This problem involves assigning start times to each operation to respect the dependencies between the gates. To efficiently program the gates, the duration of the gates, the dependencies between them and parallel execution are considered.

### 2.4 Cuccaro adder

The three main phases of compilation of quantum programs: mapping, routing and and scheduling of the gates; will be applied to the Cuccaro adder. This is a contribution to quantum arithmetic where efficient addition is performed using a ripple carry method. This implementation allows the use of a single ancilla qubit, thereby minimizing quantum hardware resources. This method is introduced by Cuccaro et al. in his article: A new quantum ripple-carry addition circuit Cuccaro et al. [2004].

The foundation of the Cuccaro adder is the carry through the majority function $\text{MAJ}(a_i, b_i, c_i)$. Cuccaro et al. [2004] propose $c_i + 1 = \text{MAJ}(a_i, b_i, c_i)$ for $i \geq 0$ and the sum $s_i = a_i \oplus b_i \oplus c_i$, which recursively computes the sum and carry operations for n-bit numbers.

The original formula for the carry operation is given by:

$c_i + 1 = a_i b_i \oplus a_i c_i \oplus b_i c_i$

And the sum bit by:

$s_i = a_i \oplus bi \oplus c_i$ for all $i < n$, and $s_n = c_n$.

### 2.5 Practical implementation

In this work, theory has been translated into practice with an implementation based on the concepts and formulas derived from Cuccaro et al. [2004]. The majority gates and UMA have been built as a fundamental components of the quantum circuit. These gates capture the core logic of the Cuccaro adder.

The $MAJ$ gate is implemented as follows:

```
maj_c = QuantumCircuit(3, name='MAJ')
maj_c.cx(2, 1)
maj_c.cx(2, 0)
maj_c.ccx(0, 1, 2)
maj = maj_c.to_gate(label='MAJ')
```

Implementation of the MAJ gate

The implementation that decomputes the majority and adds the bits, UMA gate, has used the 2-CNOT and 3-CNOT versions. This allows greater parallelization of the circuit.

```
# 2-CNOT version of UnMajority
uma2_c = QuantumCircuit(3, name='UMA2')
uma2_c.toffoli(0, 1, 2)
uma2_c.cx(2, 0)
uma2_c.cx(0, 1)
uma2 = uma2_c.to_gate(label='UMA2')

# 3-CNOT version of UnMajority
uma3_c = QuantumCircuit(3, name='UMA3')
uma3_c.x(1)
uma3_c.cx(0, 1)
uma3_c.toffoli(0, 1, 2)
uma3_c.x(1)
uma3_c.cx(2, 0)
uma3_c.cx(2, 1)
```

```
16 uma3 = uma3_c.to_gate(label='UMA3')
```

Implementation of 2-CNOT & 3-CNOT

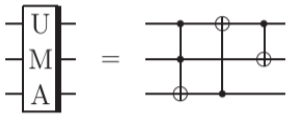


Figure 1: Theoretical implementation of Cuccaro et al. Cuccaro et al. [2004] of the 2-CNOT version

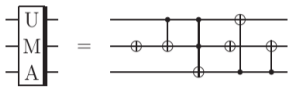


Figure 2: Theoretical implementation of Cuccaro et al. Cuccaro et al. [2004] of the 3-CNOT version

### 2.6 Quantum circuit

The Cuccaro adder circuit has been implemented by applying the MAJ gate to the input and ancillary bits. Later to the UMA gate to complete the sum. The final implementation can be seen in the figures 3 and 4

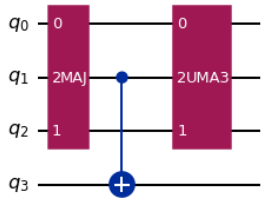


Figure 3: Implementation of the circuit with high-level gates

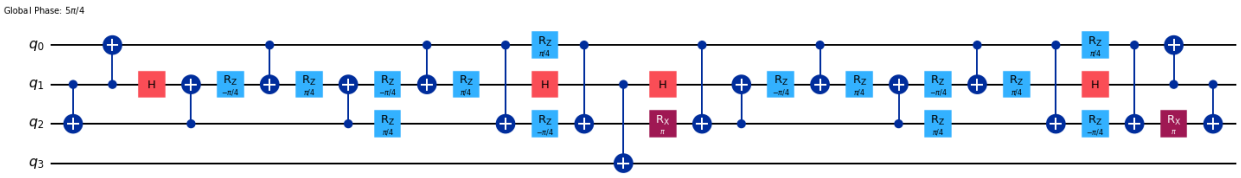


Figure 4: Implementation of the decomposed circuit in basic gates

In figure 4 a decomposition of the circuit into a set of basic gates has been carried out. This decomposition is due to the limitations of current quantum hardware. To ensure compatibility, the high-level gates MAJ and UMA have been translated into the operations supported by the QPU. QisKit's transpiler has been used to map the constructed adder circuit to the basis gates CX, RX, H, and RZ. The CX gate represents the controlled-NOT operation, RX is the rotation around the X-axis of the Bloch sphere, H is the Hadamard gate which creates superposition, and RZ is the rotation around the Z-axis.

## 3 Proposed method

To address the objectives of the notebook, which include the effective implementation of the compilation phases in a specific quantum environment, a structured method is followed that combines theoretical and experimental approaches. This method is articulated around the optimization of quantum circuits for execution on specific quantum hardware, using advanced software tools. The key aspects of this method are detailed below.

To meet the objectives of this work, a sequential implementation of the different compilation phases of a program in a quantum environment is used, in the manner described in the theoretical foundations. Advanced software tools such as Python, Qiskit, Numpy, GateWrapper and NetworkX have been used.

### 3.1 Description of Methods Used

Firstly, the implementation of the Cuccaro adder described in the theoretical foundations and which is known for its application in quantum arithmetic has been carried out. The circuit has been decomposed into basic quantum gates that are universal for quantum hardware.

Secondly, a qubit mapping has been carried out to map logical qubits to physical qubits based on the typology of the quantum hardware described. Theoretical implications have been considered, such as the objective of minimizing interactions and maintaining circuit fidelity using SWAP gates.

Thirdly, a simulation of the schedule gate has been carried out. A programming of quantum gates is developed with simulated start times of each operation.

### 3.2 Specification of Quantum Hardware Used

The quantum hardware used for the experimental execution of the optimized circuit is superconducting. An example

of such quantum hardware is the IBM Q IBM. To use this hardware, the number of available qubits, their characteristics, decoherence and error rates must be taken into account and how they can be minimized in the era of NISQ machines.

### 3.3 Software and Programming Tools

To implement the method and optimize the circuit for the specific hardware (simulated), the following software tools are used:

- Qiskit - Open source framework for working with quantum circuits and running them on simulators and real quantum hardware.
- NetworkX: Used to create and manipulate complex data structures such as graphs, which are essential for qubit mapping and routing strategies.
- NumPy: Used for numerical calculations and matrix manipulations.

## 4 Experiments and Results

### 4.1 Mapping programs to quantum hardware

The goal of mapping logical to physical qubits is to minimize the computational cost associated with interactions between qubits that are not directly connected. So this is an optimization problem where the minimization of the total distance of the shortest paths for all interactions between qubits must be found.

#### 4.1.1 Mathematical formulation

An interaction graph is defined as $G = (Q, E)$ where $Q$ is the set of qubits and $E$ is the set of edges representing interactions between qubits. Each edge $e(u, v)$ has a weight $w(u, v)$ that represents the interaction frequency between the qubits $u$ and $v$ in the quantum circuit. The distance $d(u, v)$ is the length of the shortest path between the qubits $u$ and $v$ once they have been mapped to the hardware.

The goal is to minimize the following metric:

$$\text{Minimize} \quad \sum_{u \in Q} \sum_{v \in Q} w(u, v) \cdot d(u, v)$$

subject to the constraints of the quantum hardware topology.

#### 4.1.2 Heuristic Approach

This is a type of problem widely studied in graph theory. There are many algorithms that try to provide a solution. Some of them are the Dijkstra algorithm or the Bellman - Ford algorithm. This is a type of problem with NP-hard computational complexity. So a heuristic approach can be used to find the feasible solution. The steps to follow are:

- Interaction weighting: we calculate a weight matrix $W$ where $W_{ij} = w(i, j)$ if there is an interaction between the qubits $i$ and $j$ , and $W_{ij} = 0$ otherwise.
- Distance calculation: we use the shortest path algorithms to calculate the distance matrix $D$ in quantum hardware, where $D_{ij}$ is the shortest distance between the qubits $i$ and $j$ in the hardware.
- Heuristic assignment: we proceed to map the logical qubits to the physical ones, prioritizing those pairs with greater weights to minimize the total sum of the weighted distances.

A Python function has been used that executes the previous procedure and returns a dictionary with the assignment of each logical qubit to the physical one. The dictionary has the following form:

```
Mapping: {Qubit(QuantumRegister(4, 'q'),
    1): 0, Qubit(QuantumRegister(4, 'q')
   , 2): 1, Qubit(QuantumRegister(4, 'q'
   ), 0): 3, Qubit(QuantumRegister(4, 'q
   '), 3): 2}
```

Mapping dictionary

The mapping obtained is:

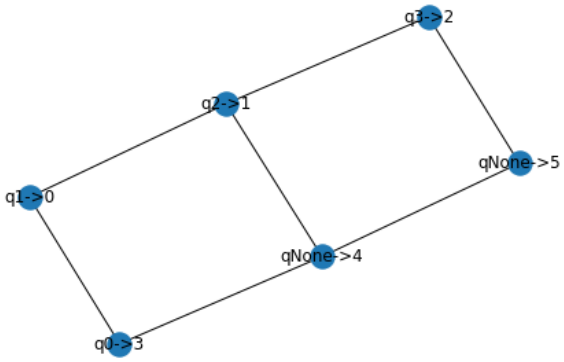


Figure 5: Graph resulting from applying the mapping with a heuristic approach

### 4.2 Routing two qubit interactions

Once the previous approach has been carried out, we proceed to route two qubit interactions. It will be considered that if two qubits are not adjacent in hardware, a SWAP will be performed to make them adjacent, the appropriate gate from the circuit will be executed and a SWAP will be performed again to return them to their original position. One of the objectives is to make the routes with the smallest number of SWAP possible.

#### 4.2.1 Mathematical representation of SWAP

The SWAP gate swaps the states of two qubits, $q_i$ and $q_j$. Mathematically, it is represented by the unitary matrix SWAP:

$$\text{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The SWAP operator is reversible and does not alter the global state of the system, except for the exchange of state amplitudes.

#### 4.2.2 Floyd–Warshall algorithm

The Floyd-Warshall algorithm computes the shortest path between each pair of vertices in a single run. The algorithm iterates through all possible intermediate vertices, updating the shortest distance between each pair of vertices, considering whether the intermediate vertex could provide the shortest path.

The shortest path matrix $D$ is updated as follows for each pair of vertices $(i, j)$, considering an intermediate vertex $k$:

$$D_{ij}^{(k)} = \min(D_{ij}^{(k-1)}, D_{ik}^{(k-1)} + D_{kj}^{(k-1)})$$

where $D_{ij}^{(k)}$ is the weight of the shortest path from $i$ to $j$ using only the vertices between 1 and $k$ as possible intermediate vertices.

For the routing of interactions of two qubits, the Floyd-Warshall algorithm is used to deterministically calculate the shortest path, since it is not an optimization problem but rather a matter of finding a fixed metric: the minimum distance for necessary exchanges.

#### 4.2.3 Quantum Routing Algorithm

To implement quantum routing theory, in practice a Python function has been created that facilitates two-qubit operations between non-adjacent qubits.

1. A new circuit is initialized with the same number of qubits as the target hardware.
2. The shortest path between every pair of hardware-grade nodes is calculated using the Floyd-Warshall algorithm. In this way, the number of minimum SWAPs for temporal adjacency is known.
3. Subsequently, it is iterated over each quantum gate of the original circuit:
   - For a single qubit the gate is applied directly to the mapped qubit.
   - For two qubits:
     - If they are adjacent in hardware, the gate is applied.
     - Otherwise, the shortest path is used to perform the SWAP operations, the gate is executed and the SWAP gates are applied in reverse order.

The adjacency achieved with SWAP is important to preserve quantum coherence and correlation between qubits.

#### 4.2.4 Optimization and Complexity

The approach used minimizes the number of SWAP operations needed to reduce quantum errors and execution times. The Floyd-Warshall algorithm has a complexity of $O(n^3)$ for $n$ qubit hardware. The more gates are applied, the greater the need and payback of applying this algorithm.

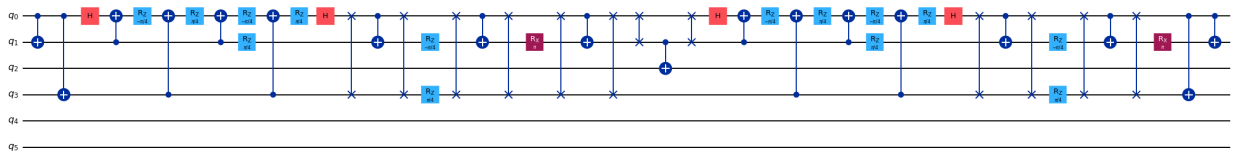

Figure 6: Routing result

### 4.3 Gate Scheduling

In real hardware, the computation of different gates entails different execution times, both at the gate level and at the level of the total quantum circuit. Therefore, to minimize delays and possible quantum errors due to decoherence, the start times of each of the gates must be scheduled.

#### 4.3.1 Mathematical foundations

As seen above, the circuit has dependencies and each node (qubit) may have to wait for some quantum dependencies. The dependency graph $G$, where each node is a quantum gate and each edge a dependency; if a gate $g_1$ must be executed before $g_2$ then there exists a directed edge $g_1 \rightarrow g_2$.

The execution time associated with each gate is denoted as $T(g)$ for a gate $g$. For the case study, the execution time

is constant $T_1$. For example, for $CNOT$ gates, the time is $T_{\text{CX}}$.

$S(g)$ represents the optimal execution time and should minimize the latest completion time of any gate, always considering the dependency constraints:

$$\min \max_{g \in G}(S(g) + T(g))$$

subject to:

$$S(g_2) \geq S(g_1) + T(g_1) \quad \forall (g_1 \to g_2) \in G$$

#### 4.3.2 Practical implementation

A Python function has been created that allows implementing the theoretical framework mentioned above. The function uses the SWAP decomposition and the dependency graph to calculate the start time as well as the maximum qubit release times. The function also accepts a parameter in the form of a dictionary, where the duration of each gate is specified.

The result is a dictionary of GateWrapper objects that represents the execution time and minimizes the total execution time while respecting the dependencies between gates. The result of the final circuit with all the phases implemented is the one in the figure 7.

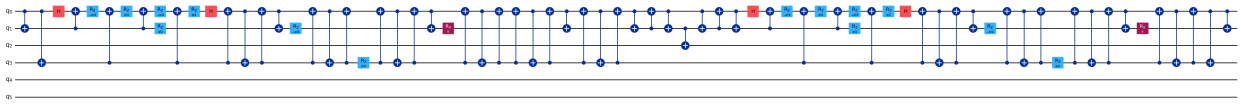

Figure 7: Final circuit after implementation of the scheduling gate

## 5 Discussion

This work has presented a structured sequence of the essential steps to compile a quantum circuit on superconducting hardware. The implementation of the techniques described in the Cuccaro adder has been explored and routing and gate scheduling have been carefully demonstrated.

The mapping of logical to physical qubits has been proven important for circuit optimization, allowing a reduction in the additional operations required. Likewise, routing is effective for the interaction between pairs of qubits that are not adjacent in hardware. Highlighting the importance of applying the minimum number of SWAP gates.

Finally, the importance of managing the execution time of each of the gates has been highlighted to adapt the circuit to parallelization processes and to minimize the quantum error; especially significant in the NISQ era. Through time management, it is possible to maximize the efficiency of the total execution time of the circuit.

## 6 Conclusions and Future Work

The application of quantum optimization techniques allows us to reduce the possibility of errors, improve circuit fidelity and better manage quantum resources. The decomposition and reconfiguration of quantum gates into basic elements has allowed the circuit created with high-level gates to be executed on current hardware.

Regarding the practical applicability of this work, the procedures and practical recommendations have been verified to improve the compilation and execution tasks of quantum circuits on real hardware. The theoretical algorithms, as well as the pseudocodes provided, offer a guide that allows direct implementation of optimization strategies.

In future work, it is expected that gate scheduling processes can be optimized to evolve in parallel with the development of more complex and hardware-demanding quantum algorithms.

The contribution of this technical report lies in presenting a comprehensible and reproducible framework on quantum circuit optimization. Valuable for researchers and practitioners in the field of quantum computing.

This report concludes the need for an in-depth understanding of quantum theory and practical implementation methodologies that often offer a multidisciplinary perspective. Involving graph theory experts and data scientists, as well as software and hardware engineers, specialists in hardware compilation.

## References

John Preskill. Quantum computing and the entanglement frontier. 3 2012. URL `http://arxiv.org/abs/1203.5813`.

Peter Wittek. *Quantum Machine Learning: What Quantum Computing Means to Data Mining*. Elsevier, 8 2014. ISBN 9780128009536. doi:10.1016/C2013-0-19170-2.

Yongshan Ding and Frederic T. Chong. *Quantum Computer Systems*. Springer International Publishing, 2020. ISBN 978-3-031-00637-1. doi:10.1007/978-3-031-01765-0.

Steven A. Cuccaro, Thomas G. Draper, Samuel A. Kutin, and David Petrie Moulton. A new quantum ripple-carry addition circuit, 2004.

IBM. Ibm quantum computing | technology. URL `https://www.ibm.com/quantum/technology`.