UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Anabel Ovide González

Multi-core Quantum Computers: Analyzing the mapping of quantum algorithms

Master's Thesis (30 ECTS)

Supervisor(s): Dr. Carmen G. Almudever
Dr. Dirk Oliver Theis

Tartu 2022

Multi-core Quantum Computers: Analyzing the mapping of quantum algorithms

Abstract:
Intermediate scale quantum computers already exist, but they are still far from showing their full computational power as they are limited by the number of qubits and noise. Therefore, a current and relevant challenge is to scale up quantum processors to build a large-scale and fault-tolerant quantum system. One of the most promising approaches for scaling quantum computers is to use a modular or multi-core architecture, in which different quantum chips are connected via quantum and classical links. A crucial aspect in multi-core quantum processor architectures is the communication between cores when qubits need to interact, as these communications are very costly. An approach to reduce these inter-core communications is by optimally mapping the quantum circuit into the quantum hardware. So far, only one mapping algorithm for multi-core quantum processors has been proposed, which is based on the relaxed Overall Extreme Exchange (rOEE) algorithm. In this thesis, we explore and analyze the application of the rOEE for mapping quantum algorithms in this kind of architecture with the aim of identifying possible limitations.

Keywords:
Quantum Computing, multi-core quantum computing architectures, mapping of quantum algorithms.

CERCS:
P170 Computer science, numerical analysis, systems, control.

Mitmetuumalised kvantarvutid: kvantalgoritmide kaardistuse analüüsimine

Lühikokkuvõte:
Mastaabitava kvantarvuti vahevariandid on juba olemas, aga need pole veel kaugeltki valmis näitama oma täielikku arvutusvõimsust, sest piiranguks on kvantbittide arv ja müra. Seetõttu on aktuaalne ja pöördelise tähtsusega ülesanne kvantprotsessorite mastaapimine, et ehitada mastaapne ja hea rikketaluvusega kvantsüsteem. Üks kõige paljutõotavam kvantarvutite mastaapimise kontseptsioon on moodul- ehk mitmetuumalise arhitektuuri kasutamine, mille korral ühendatakse kvantkiibid kvant- ja klassikaliste seoste kaudu. Mitmetuumaliste kvantprotsessorite arhitektuuri ülitähtis aspekt on tuumadevaheline side, kui tekib vajadus kvantbittidevaheliseks andmevahetuseks, sest selline side on väga kulukas. Sellise tuumadevahelise side vähendamise kontseptsioon näeb ette kvantlülituste optimaalset kaardistamist kvantriistvaras. Seni on mitmetuumaliste kvantprotsessorite jaoks välja pakutud ainult üks kaardistusalgoritm, mis põhineb rOEE (relaxed Overall Extreme Exchange) algoritmil. Selles töös uurime ja analüüsime rOEE rakendamist kvantalgoritmide kaardistamiseks sellises arhitektuuris eesmärgiga tuvastada võimalikud piirangud.

Võtmesõnad:
Kvantarvutus, mitmetuumalise kvantarvutuse arhitektuurid, kvantalgoritmide kaardistamine.

CERCS:
P170 Arvutiteadus, arvutusmeetodid, süsteemid juhtimine (automaatjuhtimisteooria).

Contents

1 Introduction

In this chapter, the motivation and the problem definition of the mapping of quantum algorithms in multi-core quantum processors will be discussed, and a structure of the thesis will be provided.

1.1 Motivation

Classical computers were born in the 1930s. Since then, they have rapidly progressed to today's technology, going from machines that need an entire room to operate to be able to fit in our hands. Despite this quick evolution, they will never be able to perform specific computations, for example, simulate Nature perfectly, as Richard Feynman stated [1].

Meanwhile, in the 1920s, quantum mechanics was discovered. With it, a new idea arises, the idea of building a quantum computer. This machine will be based on quantum phenomena such as superposition and entanglement, being able to perform computations that classical computers cannot. The first quantum computer (2-qubits) was built in 1998 by Isaac Chuang [2]; although it could only solve trivial problems, it was able to demonstrate the principles of quantum computing.

Since then, quantum computers have evolved, reaching 100 qubits, but many problems remain to be solved. One of the main issues is scalability; that is, developing a quantum system that integrates increasing qubits counts (number of qubits). A promising approach for scaling up quantum computers is to use a multi-core or modular architecture in which several quantum processors are connected via coherent quantum links and classical communication channels [3],[4],[5]. This architecture has the potential to unlock the scalability issue, but it leads to new challenges.

1.2 Problem statement

Current quantum processors, known as Noisy Intermediate-Scale Quantum (NISQ) processors [6] present different restrictions, such as the limited connectivity between qubits, the type of operations that can be performed, or their error-proneness. These limitations make that quantum algorithms, usually described as quantum circuits, cannot be directly executed on quantum devices (as they are programmed), but they have to be modified to meet the processor constraints. This process in which the quantum circuit suffers some changes is done by the compiler and is known as the mapping of quantum algorithms.

One of the main constraints in current quantum chips is the limited connectivity between qubits, which restricts the possible interactions between them (e.g. nearest neighbor interaction). Usually, qubits can only interact with adjacent qubits. Thus, when two qubits that do not share a 'connection' have to perform a two-qubit gate (interact), they need to be moved to adjacent positions by means of, for instance, SWAP operations.

The overhead in terms of time and number of extra operations this task (routing of qubits) results in, is important to consider since qubits decohere and gates are prone to errors, decreasing the algorithm success rate.

There are several algorithms that have been proposed to overcome this problem in current single-core quantum computers [7], [8], [9]. However, a new problem arises in multi-core architectures: the movements between cores. Those movements are expensive, so they must be considered when doing the mapping trying to reduce them; also, the time it takes to move the qubits between cores might be not deterministic. So far, only one mapping algorithm for multi-core quantum computing architectures has been proposed, which is based on the relaxed Overall Extreme Exchange (rOEE) [10]. This algorithm has only been tested in a single and fixed architecture consisting of 10 cores with ten qubits each and under quantum algorithms (benchmarks) which number of qubits ranges form 50 to 100. In this thesis we will further analyse the performance and identify possible limitations of the application of the rOEE algorithm to the mapping of quantum circuits in multi-core quantum computing architectures. Two main metrics will be used: i) number of non-local communications it results in, and ii) execution time (time it takes to find a solution). To this purpose, we will change different architectural parameters such as the number of cores and qubits per core as well as algorithm characteristics such as gate density. In addition, a scalability study will be performed.

1.3 Thesis structure

The thesis organization is as follows:

- Chapter 2 provides a brief introduction to quantum computing. Also, the different quantum technologies and the scalability problem are discussed, providing an introduction to multi-core quantum computing architectures (distributed architectures).
- Chapter 3 explains the mapping process in single-core and multi-core quantum computing architectures, providing examples. Also the relaxed Overall Extreme Exchange algorithm (rOEE) algorithm will be introduced.
- Chapter 4 discusses the benchmarks and the framework developed to run the experiments of this thesis. The main contributions are also presented.
- Chapter 5 shows the experiments performed using the rOEE algorithm, explaining in detail the significance of the results.
- Chapter 6 presents the conclusions of this thesis, and proposes future lines of work.

2 Background

This chapter will introduce the basics of quantum computing, what qubits are, essential quantum gates, and how a quantum circuit is represented. In addition, an overview of the current status of quantum computing will be provided, and current challenges will be discussed, emphasizing the scalability issue and possible proposed solutions, including the use of multi-core or distributed architectures.

2.1 Quantum Computing

Quantum computers are a new revolutionary technology that will be able to solve intractable problems for classical computers, thanks to the use of quantum mechanics phenomena such as superposition and entanglement.

Examples of problems that quantum computers will be able to solve include optimization, quantum simulation, and search [11]. Some of the current application fields are quantum machine learning, computational chemistry and cryptography.

For instance, the Rivest-Shamir-Adleman (RSA) algorithm is used to encrypt data securely. RSA is based on factoring large integers, which is computationally intractable for classical computers, but quantum computers could easily crack this cipher using the Shor's quantum algorithm. Currently, quantum computers are not large enough to crack the RSA cipher, but it might be possible in the future.

In this section, we will discuss the fundamentals of quantum computing.

2.1.1 Qubits: the unit of information

The basic unit of information in quantum computing is a quantum bit or qubit, which can be implemented as the spin of an electron or the polarization of a photon, a two-level physical system. The main difference between bits and qubits is that they can be in both states 0 and 1 simultaneously. This phenomenon is called superposition and it is one of the main characteristics of quantum computing.

The quantum states 0 and 1 are represented by the Dirac's bra-ket notation as $|0\rangle$ and $|1\rangle$, respectively. The state of a qubit can be mathematilcally described as:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \qquad \alpha, \beta \in \mathbb{C} \tag{1}$$

where $|\psi\rangle$ represents a quantum state, and α and β are the amplitudes.When reading out or measuring a qubit, we will get a binary result, 0 or 1, collapsing the quantum state to $|0\rangle$ or $|1\rangle$ and losing the information about α and β. In this equation $|\alpha|^2$ is the probability of measuring 0 and $|\beta|^2$ is the probability of measuring 1. Since $|\alpha|^2$ and $|\beta|^2$ are probabilities, the sum of them must be one.

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2}$$

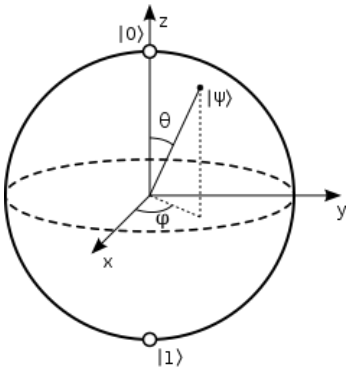


Figure 1. The Bloch sphere for representing single quantum states.

Equation 1, that is the state of a qubit, can also be represented by a vector,

$$|\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{3}$$

where the coefficients α and β can be defined as $\alpha = \cos\frac{\theta}{2}$ and $\beta = e^{i\phi}\sin\frac{\theta}{2}$ being θ and ϕ real numbers [12]. This gives us the opportunity to represent it graphically in the so called Bloch Sphere, as shown in Figure 1. The Bloch sphere has radius one, and quantum states $|0\rangle$ and $|1\rangle$ can be represented by an arrow (vector), pointing to the north and south pole of the sphere, respectively. Since qubits can be in superposition and are normalized (4), they can be at any position on the Bloch sphere surface.

We are not limited to only using one qubit, we can use several qubits. A quantum state with n qubits will have a 2^n size and is represented as follows.

$$|\Psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle = |\psi_1\psi_2 \ldots \psi_n\rangle \tag{4}$$

For example, a system composed of two qubits is shown in Equation 5. It consists of the states $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$ with the corresponding α amplitudes. Note again that $|\alpha_1|^2 + |\alpha_2|^2 + |\alpha_3|^2 + |\alpha_4|^2 = 1$.

$$|\Psi\rangle = \alpha_1 |00\rangle + \alpha_2 |01\rangle + \alpha_3 |10\rangle + \alpha_4 |11\rangle \tag{5}$$

In addition to superposition, there is another important quantum property, called entanglement. Entanglement happens when qubits are strongly correlated, and qubit's states cannot be described independently. That is, states cannot be expressed separately in tensor products. An example of an entangled quantum state is shown in Equation 6, representing a Bell State.

$$|\Psi\rangle = \frac{1}{\sqrt{2}}\big(\,|00\rangle + |11\rangle\,\big) \tag{6}$$

2.1.2 Quantum gates

As logic gates are for conventional computing, quantum gates are for quantum computing. Qubits states can be changed to other states by quantum gates. After applying a gate, the quantum state needs to be valid, meaning that it must preserve the normalization.

Since quantum states need to be valid, the gates need to satisfy some requirements:

- Gates must be unitary matrices. $U^\dagger U = UU^\dagger = I$.
- Quantum gates must be square matrices of size $2^n \times 2^n$, where n is the number of qubits involved in the gate operation.

We can differentiate two types of gates: single-qubit and multi-qubit gates. A single-qubit gate is a unitary square matrix with size 2×2, and can be interpreted as a rotation around one of the axis of the Bloch Sphere. The most important single-qubit gates are the ones called Pauli gates. They rotate 180^o around the x-, y-, and z-axis, and their names are X, Y, and Z gate, respectively. Another example of a single-qubit gate is the Phase (S) gate that performs a 90-degree rotation around the z-axis. A crucial gate in quantum computing is the Hadamard gate. This gate does a 180^o rotation over the x- and z- diagonal axis, creating a superposition of states $|0\rangle$ and $|1\rangle$. In Table 1,these gates, their symbols, and their matrices are shown.

I	X	Y	Z	H	Phase S
I	X	Y	Z	H	S
$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$

Table 1. Most relevant single-qubit gates.

When talking about multi-qubit gates, the most popular ones are the two-qubit gates. As its name suggests, two-qubit gates perform an operation in which two qubits are involved. The controlled two-qubit gates operate only in one qubit, depending on the state of the other. For example, the Controlled-NOT gate or CNOT gate uses one qubit as control, marked with $\bullet$, and the other as the target represented with $\oplus$; if the control qubit is $|1\rangle$, it will flip (apply an X gate) the target qubit, and if the control is $|0\rangle$, the target qubit will not change. Another two-qubit gate is the Controlled-Z or Control-Phase gate. It performs a Z Pauli gate on the target qubit if the control qubit is $|1\rangle$, expressed in the circuit with the black dot as mentioned before, and the target represented with the gate symbol. Another important two-qubit gate that is not a control-target gate is the SWAP gate, which exchanges the state of two qubits. This gate is essential in the mapping

process that we will see in the next chapter. In general, two-qubit gates are crucial in a quantum computer since they can create entanglement between qubits. In Table 2 we can find the most important two-qubit gates with their symbols and corresponding matrices.

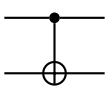
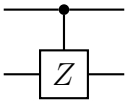
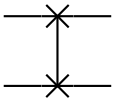
CNOT	CPHASE	SWAP
		
$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Table 2. Most popular two-qubit gates.

Like in classical computers, a universal set of quantum gates can be defined for quantum computers. A universal quantum gate set is a set of gates to which any other unitary operation can be expressed as a finite sequence of gates. A set example can be the Pauli gates (X, Y, Z), the phase shift S gate, and the CNOT.

2.1.3 Quantum circuits

Quantum algorithms are usually described as quantum circuits. They consist of qubits, gates, and measurements.

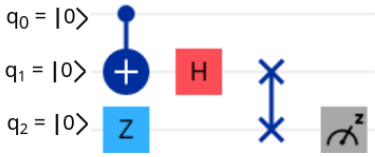


Figure 2. Quantum circuit example from IBM Quantum Experience [13].

In Figure 2 an example of a quantum circuit is shown. Circuits are read from left to right, where qubits with their initial quantum states are on the left side. The example circuit consists of three qubits, q_0, q_1, and q_2. On the right side, there is a measurement box in q_2. Until the measurement, the gates we can find are a CNOT between q_0 and q_1, a Pauli Z in q_2, a Hadamard in q_1, and a SWAP between q_1 and q_2.

Finally, some properties of the circuit that are relevant for the compilation or mapping process that will be explained later are:

- Latency or execution time: the time the circuit execution takes.
- Circuit depth: the number of total time steps of a circuit, where a time step is the set of operations that can be executed in parallel at a specific time in the circuit.
- Circuit width: the total number of qubits.

2.2 Quantum computers nowadays

Quantum computers have experienced a remarkable progress in the last few years, moving from theory to reality, and from a few qubits to tens of qubits. However, there is a long path to go as we can consider that quantum computers are still in their infancy.

In this section we will discuss the different quantum computing technologies, current quantum processors, and the scalability issue, one of the most significant quantum computer challenges.

2.2.1 Current quantum technologies

Current quantum computers are also know as NISQ (noisy intermediate-scale quantum) [6] computers. Intermediate-scale refers to the qubit count or number of qubits which is currently limited to a few tens. Because of this limit in the number of qubits, it is challenging to implement quantum error correction since this technique requires a considerable amount of qubits. Noise refers to unwanted interactions between qubits and the environment (decoherence) and the high gate error rate.

There are several quantum technologies for qubit implementation, a summary of them is shown in Table 3. The most important ones are:

- Superconducting. They work with currents that oscillate in a circuit loop [14]. To perform gates, they use microwave signals. The problem with this technology is the short qubit coherence time, and they must be kept cold. The largest superconducting quantum computer consists of 127 qubits, the IBM Eagle quantum processor [15].
- Ion trap. Physical qubits are confined using electromagnetic fields [16]. Lasers do the cooling of the system and the coupling between qubits. This technology is the most promising one in terms of scalability, but the operations are slow, and many lasers are needed. Ion trap based quantum computers developed by Honeywell have achieved a Quantum Volume[1] of 128 [17].

[1]Quantum Volume quantifies the largest square circuit that the quantum computer can successfully execute

- Diamond vacancies. A nitrogen atom with a vacancy in the center adds an electron to a diamond lattice, creating a qubit [18]. These qubits are controlled with light and can operate at room temperature, but they are sensitive to external electrostatic fields.

- Quantum dots. They are semiconductor nanoparticles made by adding an electron to a tiny piece of silicon, and microwaves control the quantum state [19]. They are very stable but must be kept cold. Also, it is challenging to perform entanglement with high fidelity.

- Neutral atoms. Ground states of a neutral atom (Rb, Cs) create a qubit. As the diamond vacancies, these atoms are controlled with light, but it is needed to achieve more advanced atom cooling to improve the fidelity [20], [21].

Nowadays, the most promising quantum technologies are superconducting and trapped-ion since they have achieved the highest qubit counts. Other relevant technologies are NV centers, and silicon quantum dots. To understand the quality of a quantum computer,it is important to look at the following properties:

- **Number of physical qubits:** The more physical qubits a quantum computer has, the higher its computational power. In addition, if we have a considerable number of physical qubits, it will be possible to use quantum correction techniques.

- **Qubit lifetime or coherence time:** How long it takes qubits to be degraded by the environment. It will be explained in more detail in the next section.

- **Gate operation time:** The time it takes to perform a gate operation.

- **Gate fidelity:** It indicates the accuracy/reliability of a quantum gate operation.

- **Connectivity:** How qubits are connected. As we will explain in more detail, it determines the possible interactions between qubits.

Some of today's most advanced quantum processors are the IonQ Aria [17] by IonQ and the Eagle [15] by IBM. The IonQ Aria quantum processor is based on ion trap technology. IonQ claims that the IonQ Aria is the most powerful quantum processor in the industry, achieving a record of 20 algorithmic qubits (QA) [23], a new metric by IonQ based on structured quantum circuits rather than random quantum circuits as Quantum Volume.

The Eagle is a 127-qubit quantum processor being the first quantum computer able to go beyond the 100 qubits. It is based on a heavy-hexagonal [24] qubit layout. Since this is such a new quantum processor, there are no metrics of its Quantum Volume performance or gate fidelities yet.

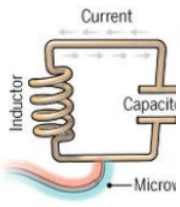
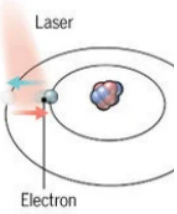
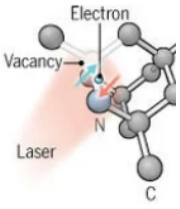
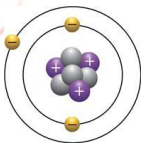
Quantum technology	Superconducting	Ion trap	Diamond Vacancies	Quantum Dots	Neutral atoms
Qubit encoding description	Two-level system of a superconducting circuit	Electron spin direction of ionized atoms in vacuum	Occupation of a waveguide pair of single photons	Nuclear or electron spin or charge of doped P atoms in Si	Hyperfine atoms ground states
Qubit lifetime	$50 - 100\mu s$	$50s$	$150\mu s$	$1 - 10s$	$100\mu s$
Gate fidelity	99.4%	99.9%	98%	90%	97%
Gate operation time	$10 - 50ns$	$3 - 50\mu s$	$1ns$	$1 - 10ns$	$7ns$
Connectivity	Nearest neighbors	All-to-all	To be demonstrated	Nearest neighbors	All-to-all
Scalablility	No major road-blocks near-term	Scaling beyond one trap(> 50 qb)	Single photon sources and detection	Novel technology potentially high scalability	Advanced cooler atom cooling is required
Example image					

Table 3. Comparison between different quantum technologies. Data from [22], [20], [21]. Images credits to (GRAPHIC) C. Bickel/Science;(DATA) Gabriel Popkin.

QuEra plans to launch this year a new quantum technology based on neutral atoms (Rydberg atoms) [25]; it will have 256 qubits, surpassing IBM's Eagle quantum processor.

Note that to evaluate the performance of a quantum processor, IBM has recently defined three key factors [26]: quality, speed, and scale. Quality is defined by Quantum Volume, indicating how accurately a quantum circuit can be performed. It is sensitive to coherence, gate fidelity, measurement fidelity, and connectivity. Speed is the number of Quantum Volume circuits the quantum processor can execute per unit of time, measured in CLOPS (Circuit Layer Operations per Second). Scale is given by the number of qubits and determines the computational power of the quantum processor.

Therefore, scaling up quantum computers is key for achieving their execptional potential. However, they all share the same challenge: overcoming the considerable error rate as the number of qubits increases.

2.2.2 Scalability problem description and solution proposals

Scalability is necessary to achieve a general-purpose quantum computer, but to go beyond the NISQ era and develop a scalable quantum computer is a challenge nowadays since it must handle an increasing number of qubits without losing fidelity.

The most challenging part of scalability is the qubit nature. Qubits are fragile and prone to errors. They are incredibly delicate as their state can be easily modified due to their interaction with the external environment. This behaviour in which qubits lose their information is called decoherence, as shown in Figure 3. In addition, gates and measurements are implemented with imperfect and error-prone operations. Another source of errors is crosstalk [27] which is caused by undesirable interactions between coupled qubits. All these errors scale as the number of qubits increases.

This scalability of errors as the number of qubits increases makes very challenging to scale up current quantum processors that follow a monolithic approach; the more qubits there are on a single chip and the more gate operations we perform, the more error-prone our quantum computer will be.

A distributed or multi-core quantum computer has been proposed to overcome the scalability problem. The main idea is to scale quantum computers without increasing noise or crosstalk by having several chips connected between them via quantum coherent links as well as classical links. There have been proposals for quantum intranet technologies [28], distributed superconducting quantum technologies [3] and distributed quantum computers based on trapped ions [4]. With this approach, new challenges emerge that need to be addressed, such as performing an efficient compilation [29] and communication between cores [30].

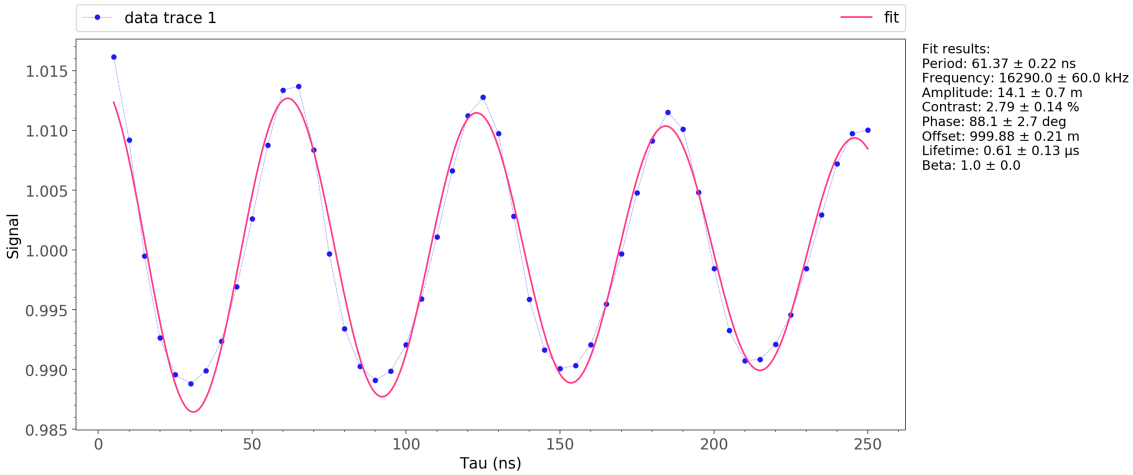


Figure 3. A Rabi experiment in an NV-center is shown; the waves a qubit produces in a real experiment. These waves represent a quantum state change over time due to decoherence—the wave amplitude changes in time due to the interaction with the environment. The wave will maintain its amplitude in a perfect scenario, having no change.

2.3 Multi-core Quantum Computers

Multi-core quantum computers or distributed quantum computing architectures seem to be a promising approach towards scalability. Existing proposals [3] [4] [28] agree to use an architecture based on interconnecting (short-range quantum-coherent interconnections) multiple NISQ quantum processors, ranging from tens to hundreds of qubits. A representation of this multi-core computer can be seen in Figure 4. With qubits distributed among cores instead of having all of them on a single-core, this processor improves isolation, decreasing the number of possible errors. It also has the advantage of parallelism, as simultaneous operations can be performed on different cores at the same time.

The connections between cores can be implemented by ion shuttling, photonic switches, or quantum teleportation. A proposed layered architecture of a multi-core quantum processor [31] is shown in Figure 5 The main difference with a single-core architecture relies on the network layer and the communication process since different cores must be connected and exchange information.

One problem arises with this new architecture: exchanging information among cores. In quantum physics, there is a theorem called *"no-cloning theorem"*, which states that it is impossible to clone information from one qubit to another. Due to this restriction, the communication between cores requires that qubits need to move from one core to another by means of, for instance, shuttling or teleportation. These movements are more costly than the movements required in a single-core quantum computer, so we must try

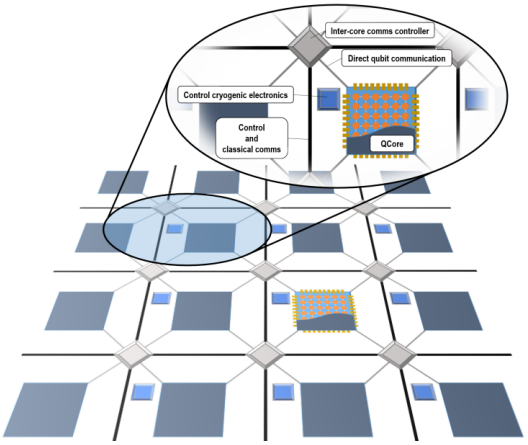


Figure 4. Multi-core quantum processor. Image taken from [31].

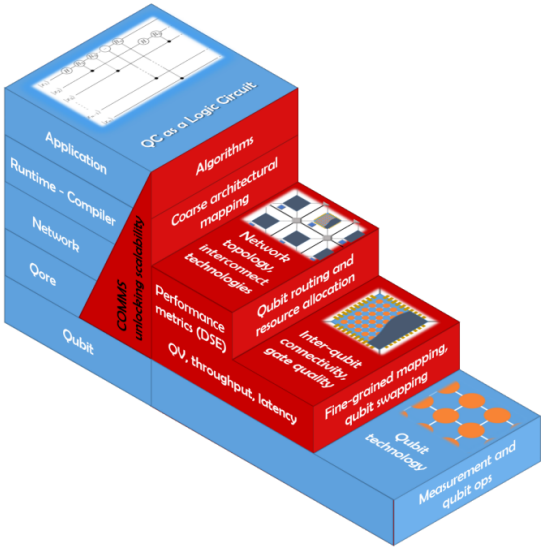


Figure 5. Full-stack layered architecture for multi-core quantum computers. The red part corresponds to the layers that are required for managing communication between cores. Image taken from [31].

to reduce these movements between cores as much as possible.

Therefore, efficient mapping (compilation) circuit techniques need to be developed to execute a quantum algorithm on a multi-core quantum computer. In the next chapter, the mapping of quantum algorithms will be introduced for single-core and multi-core quantum processors.

3 The mapping of quantum algorithms

Quantum algorithms cannot be directly executed on quantum hardware, as each quantum processor has a series of constraints. The main restriction is the limited connectivity between physical qubits; quantum technologies such as superconducting and quantum dots need qubits to be adjacent to perform two-qubit gates, but quantum circuits assume qubits have an all-to-all connection. The mapping of quantum algorithms (compilation process) solves that restriction, adapting a quantum circuit to the requirements of each quantum processor, being one of the most critical steps when executing a quantum algorithm.

This chapter will describe the mapping process and its stages in a single-core quantum computer and compare it with a multi-core quantum computer, showing some illustrative examples and focusing on the only mapping algorithm that has been proposed so far for multi-core quantum processors, which is based on the Relaxed Overall Extreme Exchange algorithm [10].

3.1 Executing algorithms in a single-core quantum computer

Current monolithic quantum processors present different constraints that must be considered when executing a quantum algorithm (described as a quantum circuit) on them. The main quantum processors constraints are: i) elementary gate set; that is, the gates supported by a given quantum device, and ii) qubit connectivity or how qubits are arranged and connected in the quantum processor, which limits their possible interactions (i.e. two-qubit gates). These constraints require quantum algorithms to be modified to run on a given quantum processor. The process that adapts the quantum circuit to the restrictions of each quantum processor is known as mapping.

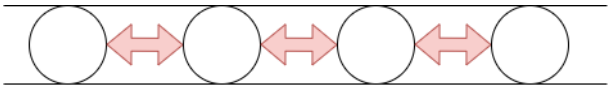


Figure 6. An example of a quantum processor in which qubits are arranged in a 1-dimensional array. Circles represent the qubits and the arrows the connections between them. Qubits have to be adjacent to perform two-qubit gates.

The mapping procedure consists of four main steps: gate decomposition, initial placement, routing, and scheduling. These steps take place in the compiler of the quantum computer but not necessarily in this order. An example can be seen in Figure 7. When explaining the steps, some examples will be provided based on the quantum processor shown in Figure 6. In this quantum processor, qubits will be arranged in a 1-dimensional (i.e. linear) array, so to perform two-qubit gates, the involved qubits must be adjacent, needing SWAPs operations to be moved.

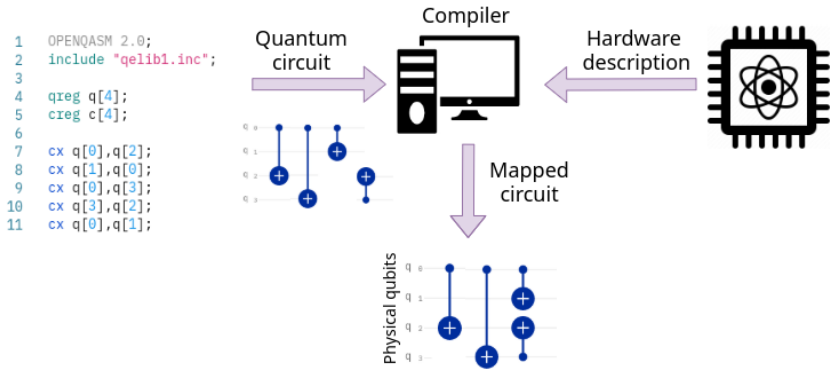


Figure 7. The mapping process flow. The code of the quantum algorithm to be executed (on the left) together with the hardware architecture/constraints description are provided as inputs to the compiler. With this information, the compiler decomposes the gates into ones the quantum processor support. It also schedules the operations and performs the qubits initial placement and the routing process.

Not all quantum processors support the same quantum gates, but all of them must support a set of universal quantum gates and be able to perform two-qubit gates to produce entanglement. As mentioned in the second chapter, all circuits can be represented as a set of universal quantum gates that can be executed in any quantum processor. The process of decomposing the gates of the quantum circuit to the ones supported by the quantum hardware is called **gate decomposition**.

The **initial placement** is responsible for placing the logical qubits of the quantum circuit into the physical ones in the hardware, trying to benefit further steps minimizing future SWAPs operations needed in the routing stage. This step is especially beneficial for short algorithms since the first qubits position can significantly reduce the total SWAPs. Note that the initial placement is a NP-complete problem [7]. An example of the benefits that the initial placement can bring is shown in Figures 8 and 9. In both figures, an initial placement of qubits (optimal and trivial) for the same quantum circuit is performed. Figure 8 shows how an optimal initial placement can save up to 4 SWAPs in further steps as we will discuss in the next paragraph.

After the initial placement step, the **routing** process checks whether all two-qubit gates can be executed. When a two-qubit gate cannot be performed, that is, when the two qubits of the gate are not nearest neighbors, they need to be moved to adjacent positions, usually by means of SWAP gates. In this case, the routing finds a path (e.g. shortest path) and introduces the required SWAP gates. The routing process plays a key role in the mapping procedure, ensuring that all quantum gates can be performed so that the quantum circuit can be executed in the quantum computer. It should be noted that the

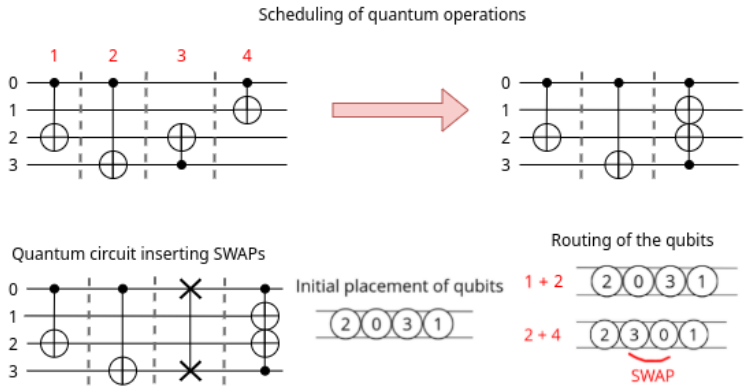


Figure 8. Optimal mapping of the logical qubits to the physical qubits of the quantum processor shown in Figure 7. In total 1 SWAP has to be added for being able to perform all two-qubit quantum gates.

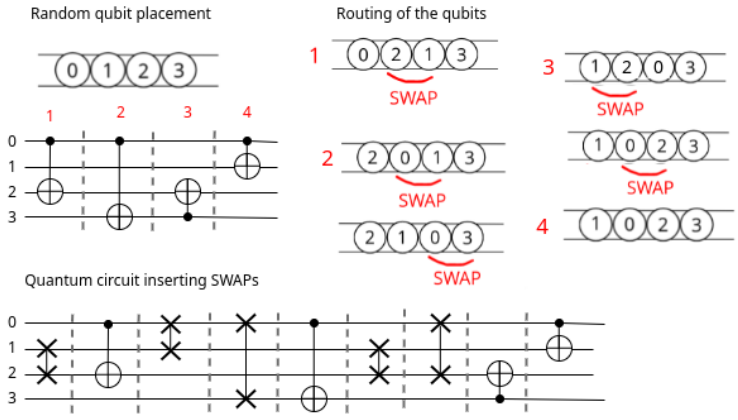


Figure 9. Trivial mapping of the logical qubits to the physical qubits of the quantum processor shown in Figure 7. In total 5 SWAPs has to be added for being able to perform all two-qubit quantum gates.

routing step results in an increase in the number of gates and the circuit depth, decreasing the success probability of the algorithm.

Routing examples are shown in Figures 8 and 9. To perform the routing in those examples, the quantum processor (Figure 6) constraints must be taken into account: qubits need to be adjacent when a two-qubit gate is executed, to do so, SWAPs operations must be performed between the nearest neighbors qubits. In Figure 8, for the first two two-qubit gates (first time step) is not necessary to insert any SWAP operation since qubits are already adjacent. However, a SWAP operation will be necessary for the next two-qubit gates (second time slice) to place qubits in adjacent positions. Note that in Figure 8, when using an optimal initial placement, the number of SWAPs inserted by the

routing process is less than in Figure 9.

The **scheduler** organizes the gates through time, parallelizing the operations while respecting the quantum processor constraints and the gate dependencies. A scheduling example is shown in Figure 8. The first two CNOTs must be serialized as there is a dependency between them (i.e. they share the control qubit). The other two CNOTs can be performed in parallel. It can be seen how a time step can be saved by parallelizing operations, increasing the probability of executing the algorithm successfully.

Most of current solutions focus on NISQ devices, particularly superconductor quantum computers. The mapping process is an NP-complete problem [7]. Therefore, brute-force approaches [32] can only be applied for a small number of qubits. For a considerable amount of qubits, heuristics solutions [33] have to be used. Some used methods are Mixed Integer Linear Programming solvers [8], Satisfiability Modulo Theory solvers [34], heuristic search algorithms [9], decision diagrams [35], and reinforcement learning [36].

3.2 Mapping in multi-core quantum computers

Multi-core quantum architectures have been recently proposed to scale up quantum computers [3], [4]. Therefore, their mapping process needs to be explored deeply, being more complicated than in single-core quantum processors due to inter-core communications. The main differences and current mapping algorithms for the multi-core architecture will be discussed, specifically the relaxed Overall Extreme Exchange algorithm proposed in [10].

3.2.1 Differences between single-core and multi-core mapping process

As explained in section 2.3, multi-core quantum processors consist of several quantum NISQ chips connected via quantum coherent links and classical communication channels. The main difference regarding mapping will be the communications between cores not presented in single-core quantum computers, called inter-core communications. These communications are more expensive than the communication within the core (intra-core), so they should be avoided as much as possible.

The mapping procedure has now a new task, optimally planning and moving qubits among cores. After that, the mapping approaches for single-core processors can be applied separately to each core so that qubits will be nearest neighbour. Several differences in the mapping steps between single-cores and multi-cores quantum architectures can be noted.

The **scheduler** process will have the same function as in single-core quantum computers, but in this case, it will be more complex due to inter-core communications. The scheduler must be dynamic since the time the communication operation takes might be unknown at compile time. The time consumption of qubit movements between cores is

not deterministic (e.g. teleportation), so the time is calculated at runtime. In addition, the scheduler must have control of the available resources, i.e. which qubits are avaialable and which ones are occupied to perform inter-core movements.

The **initial placement** must consider that not all qubit movements will have the exact cost since inter-core movements will be more expensive than intra-core movements. It should place the logical qubits into the physical ones, considering the cost of the different movements and the qubits interactions; in other words, qubits that will interact (i.e. perform a two-qubit gate) must be placed, if possible, in the same core.

The **gate decomposition** and **routing** process will be similar to single-core architectures.

As far as we know, the only mapping technique proposed so far for multi-core or modular quantum computing architectures is the one presented in [10]. They develop a complete system for mapping quantum algorithms to near-term cluster-based quantum architectures based on graph partitioning.

Their approach bases the initial placement process on a static partitioning graph problem (with fixed-size partitions). However, there are a few algorithms that find approximate solutions [37] [38] to this problem. There are also heuristic solvers [39], but they cannot guarantee the fixed size of the partitions.

For the routing problem, posed as a qubit assignment problem, instead of finding a global assignment of the qubits to the cores, they partition the circuit into time slices (i.e. each slice consists of two-qubit gates that can be performed in parallel) and find an assignment for each of them, as shown in Figure 10. To this purpose they use the relaxed Overall Extreme Exchange (rOEE) algorithm, which will be explained in detail in the next section.

3.3 The relaxed Overall Extreme Exchange algorithm

The relaxed Overall Extreme Exchange algorithm (rOEE) [10] is at the core of the algorithms for multi-core quantum processors. All steps of this algorithm will be explained, but before, it is necessary to understand the Overall Extreme Exchange (OEE) [40] algorithm on which it is based on.

The rOEE and OEE algorithms will represent the mapping problem as a partition graph problem in which physical qubits correspond the nodes of the graph, edges will be the allowed interactions between physical qubits, and the cores the partitions. When two qubits need to interact, the edge will have a cost defined by a lookahead weight, called W. In [10] they use an all-to-all connection between cores and between physical qubits within the cores, so the same connectivity will be used in this work.

3.3.1 Description of OEE algorithm

The Overall Extreme Exchange algorithm (OEE) [40] is a natural extension of the KL algorithm. Given an initial placement of the qubits, OEE consecutively selects a pair of qubits to exchange. In this exchange, the selected qubits maximize the reduced exchange cost. The exchanges continue if the exchange cost is greater than zero, otherwise there is no more improvement. The algorithm stops when the summation of all exchange costs is less or equal to zero.

To understand the OEE algorithm the follow definitions are needed: the set of partitions will be defined as P, and a color l is the partition a node i is in. So, $col(i) = l$ if $i \in P_l$.

The detailed steps of the algorithm are[2]:

- Step 1. Construct a list C with all the nodes. Calculate $W(i,l)$ and $D(i,l)$ for each i, l, where W and D are:

$$W(i,l) = \sum_{j \in P_l} w_{ij} \tag{7}$$

$$D(i,l) = W(i,l) - W[i, col(i)] \tag{8}$$

- Step 2. Find a and b such that $g(a,b) = max_{ij} g(i,j)$, where:

$$g(i,j) = D[i, col(j)] - D[j, col(i)] - 2w_{ij} \tag{9}$$

  After that, delete a and b from C

- Step 3. Update $D(i,l)$ based in C.

- Step 4. If C is not empty repeat from Step 2.

- Step 5. Find x such that the total reduced cost is maximum adding all g's:

$$g_{max} = g_1 + g_2 + \ldots + g_x \tag{10}$$

- Step 6. Exchange the first x pairs and save the partition.

- Step 7. If $g_{max} > 0$ repeat from the first step, otherwise we have found the solution.

In summary, the OEE algorithm exchanges a pair of qubits and performs as many exchanges as it will give an overall benefit. The main problems with this algorithm are: it often overcorrects, minimizing the cost between partitions without considering the qubits interaction with the next or previous time slices, and the execution time can be high. The rOEE algorithm, an extension of the OEE algorithm, has been proposed to solve these issues.

[2]For proof of the formulas check [40]

3.3.2 Description of rOEE algorithm

To solve the problems encountered in the OEE algorithm, rOEE proposed using lookahead weights to take into account the following time slices and a relaxed version of OEE so as not to overcorrect and reduce the execution time.

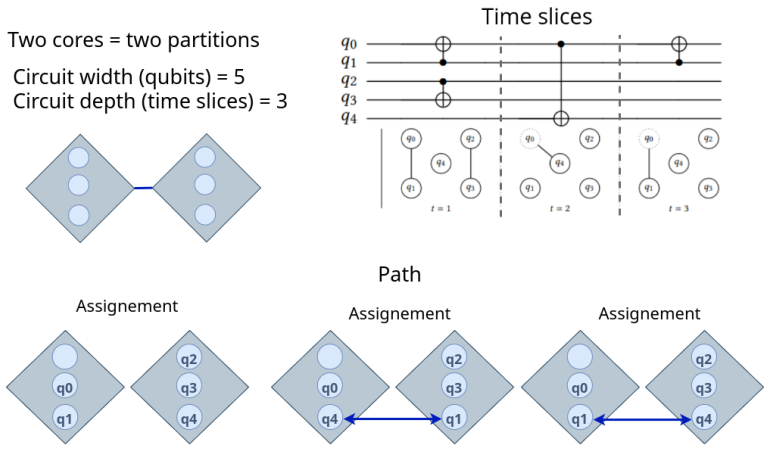


Figure 10. Example of how the rOEE algorithm works for a simple quantum circuit. An architecture consisting of two cores with three qubits per core is considered, meaning we have two partitions. We can see the path formed by valid assignments of each time slice. Quantum circuit and graphs from [10]

Some definitions are needed to understand the algorithm. An assignment is a set of qubits partitions in a specific time slice; an assignment is valid when the qubits that interact in a specific time slice are in the same partition. The path will be the sequence of valid assignments through the quantum algorithm. Figure 10 illustrates these definitions and how the rOEE algorithm works. In the first time step, two two-qubit gates between qubits q0 and q1 and q2 and q3 have to be performed. The rOEE algorithm places the qubits involved in the two-qubit gate in the same partition (core), the figure shows how the qubits q0 and q1 are in the first core, and qubits q2 and q3 are in the second core, making a valid assignment. In successive time slices, the two qubits involved in two-qubit gates will perform exchanges with other qubits until both are in the same partition. After finding valid assignments for all three time slices, the rOEE algorithm will output the path (all valid assignments).

The rOEE algorithm runs OEE until it finds a valid assignment for a time slice and makes no more exchanges. The lookahead weights will provide information about which qubits will be beneficial to exchange. The constant function, exponential decay, and gaussian decay have been tested as weighting functions in the original work [10], with exponential decay having the best results. Also, since the OEE algorithm is relaxed, it is

speeded by the rOEE.

In [10], the application of the rOEE algorithm for mapping quantum circuits into multi-core quantum computing architectures has only been tested for a single fixed architecture consisting of ten cores and ten qubits per core, i.e. 100 physical qubits, and for quantum algorithms (benchmarks) consisting from 50 to 100 logical qubits. In this thesis, we will further analyse the performance of the rOEE algorithm and identify possible limitations. To this purpose, we will try different architectures in which the number of cores and the number of qubits a per core are changed depending on the algorithm requirements (how many qubits are necessary). In addition, we will explore how it behaves when different characteristics of the quantum algorithms, used as benchmarks, are changed such as the two-qubit gate density. An scalability analysis will be also performed.

4 Benchmarks and simulation framework

In this chapter, the code, quantum algorithms (benchmarks), and simulation framework necessary to stress and to further analyze the performance of the rOEE routing algorithm will be presented. The first section will show the quantum algorithms used as benchmarks. The second section will focus on all the tools used. Some benchmarks have used the OpenQL compiler [41] to produce scheduled code, which will be discussed. Also, the different implementations of the rOEE algorithm will be presented. The last point of this chapter will describe the implemented simulation framework to perform the different experiments of this thesis.

4.1 Benchmarks

To further analyze the performance of the rOEE routing algorithm, we selected the following quantum algorithms: Cuccaro adder [42], Quantum Volume [43], QFT adder [44], and QUEKO [45]. The Cuccaro adder and QFT adder are also used in the rOEE original work [10]. We have decided to use Quantum Volume (instead of the Random Circuits) because it is a widely used benchmark for measuring the quality of a quantum computer, and it also generates the circuit with a random component. In addition, the QUEKO benchmarks have been chosen since we can change the density of the two-qubit gates of the circuit, as we will see later.

It is essential to characterize the algorithms before using them as benchmarks. Figure 11 shows:

- How the number of two-qubit gates scales with the number of logical qubits of the algorithm.
- How the average of two-qubit gates evolves as the number of qubits increases, where the average of two-qubit gates per slice is calculated by dividing the number of two-qubit gates by the number of slices the algorithm has.
- How the number of slices (circuit depth) increases with the number of physical qubits.

In addition, in Table 4, the minimum and maximum values of two-qubit gates, average two-qubit gates per slice, and number of slices (circuit depth) for the four quantum algorithms when scale them up from 50 to 100 qubits are shown. They are important to analyze the rOEE routing algorithm in the next chapter. Note that usually the more two-qubit gates the algorithm has, the more inter-core movements will be required. Furthermore, the average of two-qubit gates per slice indicates the average degree of parallelization of the algorithm; the higher the average of two-qubit gates, the more

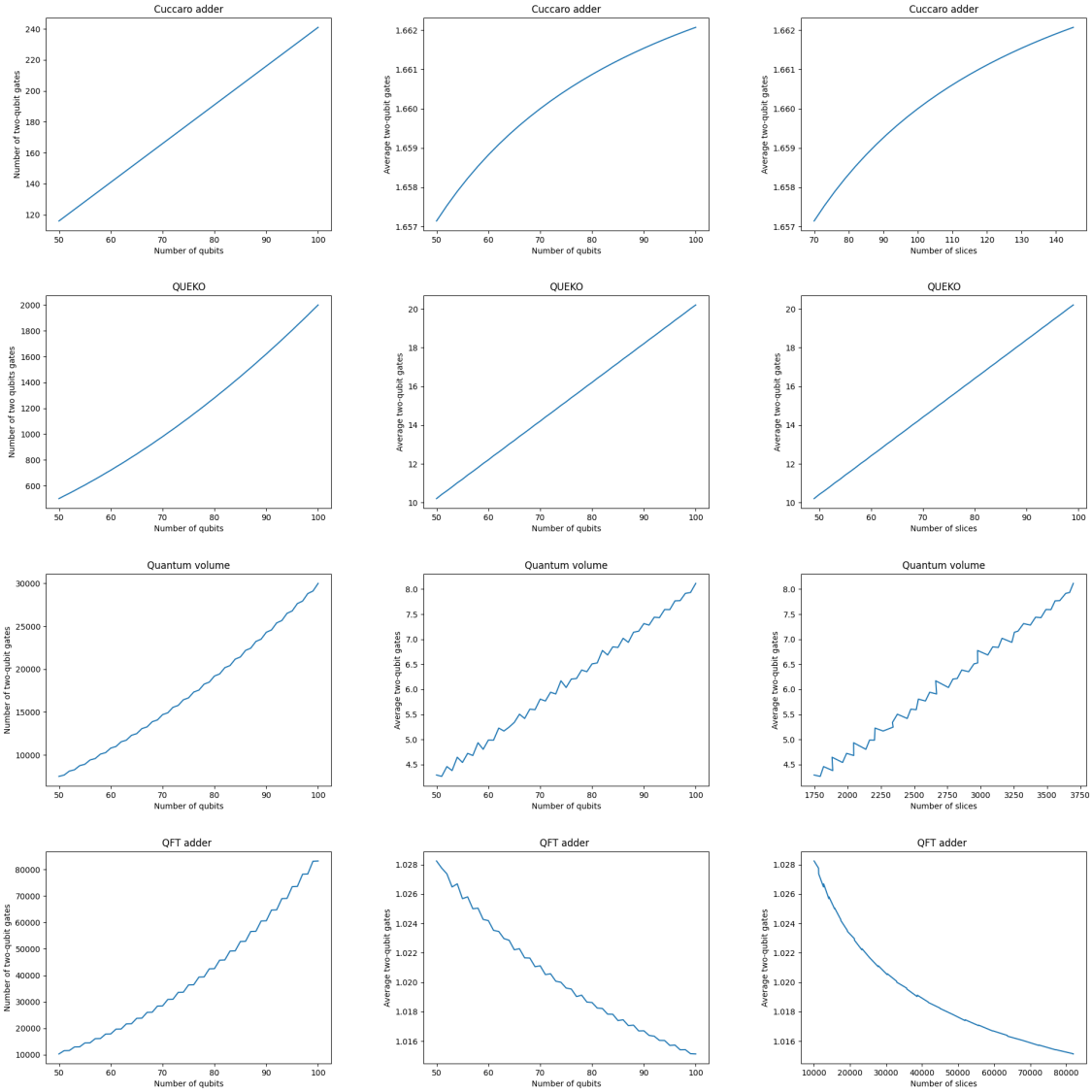


Figure 11. From the first to the last row: Cuccaro adder, QUEKO, Quantum Volume, and QFT adder. The number of two-qubit gates vs. the algorithm number of logical qubits is shown in the first column. The second column shows how the average of two-qubit gates scales vs. the number of logical qubits used. The last column shows the average of two-qubit gates vs. the number of time slices.

parallelizable the algorithm is. Finally, the more slices the algorithm has, the longer it will take to execute and the more valid assignments need to be found.

The implementation of the scalable **Cuccaro adder** used in this work is based on [42]. The most crucial points of this algorithm are the low depth and the few two-qubit gates it

Benchmark	Qubits	Two-qubit gates	Average two-qubit gates	Slices
Cuccaro adder	50-100	120-240	1.657-1.662	70-140
QUEKO	50-100	600-2000	10-20	50-100
Quantum Volume	50-100	10000-30000	4.5-8	1750-3750
QFT adder	50-100	10000-80000	1.016-1.028	10000-80000

Table 4. Minimum and maximum values of number of logical qubits, two-qubit gates, average of two-qubit gates per slice and number of slices (circuit depth) for the four quantum benchmarks.

has compared with the other selected algorithms.

The scalable **QFT adder** algorithm has been translated from C++ to Python and some gate decomposition have been performed for being able to compile the algorithm and get its scheduled code using the OpenQL compiler as it will be explained in the next section. The C++ source code is from the private repository qbench from Delft University of Technology.

The number and average of two-qubits gates, and the depth increase because of the algorithm gate decompositions. We can observe that this algorithm has a very high number of two-qubit gates and slices but a decreasing two-qubit gates average, having a low parallelizable algorithm with a high depth. We can also observe how the number of slices increases with the number of two-qubit gates and the average of two-qubit gates, having almost the same number of two-qubit gates and slices.

The **Quantum Volume** benchmark is a commonly used method to stress quantum technologies. The code used is from the private repository qbench from Delft University, and it is based on [43]. Observing the plots, it can be seen that the number and average of two-qubit gates and slices increase with the number of qubits, having a parallelizable algorithm with a considerable number of two-qubit gates.

QUEKO benchmarks are used to stress different quantum architectures [45]. The QUEKO script for building QUEKO circuits has been uploaded to the public GitHub repository[3] after being requested. A parser has been coded to translate the QUEKO script output to cQASM code. One important characteristic of this benchmark is that the density of the two-qubit gates can be changed when building the QUEKO circuits. By looking at Figure 11, we can observe that this benchmark behaves similarly to the Quantum Volume benchmark but has fewer two-qubit gates and depth. QUEKO is the most parallelizable benchmark, followed by Quantum Volume, Cuccaro adder, and QFT adder.

[3]https://github.com/tbcdebug/QUEKO-benchmark

4.2 Simulation framework

To run the experiments for the rOEE algorithm, the OpenQL framework [41] has been used to compile the algorithms and obtain and scheduled circuit in the form of cQASM code. In addition, a custom simulation framework has been implemented. This section will discuss how OpenQL works. Also, the required implementations to execute the rOEE algorithm will be pointed out. A summary of the entire simulation process is shown in Figure 12.

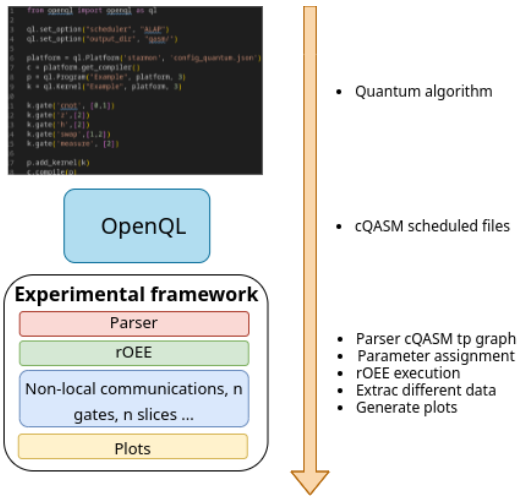


Figure 12. Simulation framework flow.

4.2.1 OpenQL

OpenQL [41] is a framework that compiles and optimizes quantum algorithms. It takes as input quantum circuits coded in Python or C++ and a JSON file used to specify the quantum architecture. The output consists of low-level instructions that can be run on different backends, including quantum computer simulators and real quantum processors. It also supports different mapping options (initial placement, scheduling, and routing). In OpenQL we can differentiate the following steps:

1. In the first step, it receives a file in Python or C++ describing the quantum algorithm.

2. Produces a cQASM file describing a quantum circuit independent of low-level hardware details.

3. Creates an optimized mapped quantum circuit taking into account the restrictions of the quantum processor.

4. Compiles the last cQASM file into low-level executable code, considering all details and constraints of the quantum processor architecture.

For this work, OpenQL has been used to provide the cQASM scheduled files with the gates divided by time slices. The Cuccaro adder, QFT adder, and Quantum Volume have been compiled in OpenQL, all of them with the 'ALAP' option when scheduling, which means the compiler starts from the last gate and schedules all gates as late as possible.

```
1  from opengl import opengl as ql
2
3  ql.set_option("scheduler", "ALAP")
4  ql.set_option("output_dir", "qasm/")
5
6  platform = ql.Platform('starmon', 'config_quantum.json')
7  c = platform.get_compiler()
8  p = ql.Program("Example", platform, 3)
9  k = ql.Kernel("Example", platform, 3)
10
11 k.gate('cnot', [0,1])
12 k.gate('z',[2])
13 k.gate('h',[2])
14 k.gate('swap',[1,2])
15 k.gate('measure', [2])
16
17 p.add_kernel(k)
18 c.compile(p)
```

Figure 13. Example of a quantum algorithm from Figure 2 coded in OpenQL.

A simple example using OpenQL is shown in Figure 13, representing the quantum circuit of Figure 2. First, we need to initialize the program and the kernel, lines 7 and 8. After it, the quantum circuit is written in lines 11 to 14. The last step is to add the kernel to the program and compile it. To execute this example, an 'ALAP' configuration has been set for the scheduling in line 3.

4.2.2 The rOEE implementation

The main rOEE implementation and input parser used in this work is the Phyton code written by a Master's degree student from the Technical University of Catalonia, who we collaborate with (Professor Eduard Alarcon's group). The rOEE Python code has been improved in this work by solving some bugs, counting the valid assignments, and improving the quality of the code by minimizing the execution time. The lookahead weight function used in the rOEE Python code is the Exponential Decay since it is the one that has been proved to work best, as mentioned in the last chapter 3.3.2.

The Python code will receive as an argument a graph as explained in the previous chapter 3.3.1. The parser will translate the cQASM code from the quantum algorithm

to the input graph. This parser has been improved by solving bugs and counting the algorithm time slices.

A python script has been coded to calculate the non-local communications produced when executing the rOEE algorithm. This script is based on the information from the rOEE paper [10].

In addition, the rOEE algorithm has been translated to C++ and integrated into OpenQL. When integrating the code, it has been necessary to learn how OpenQL works internally, with a graph-based time that considers the gates that have been already performed and future gates. Also, it has been necessary to get familiarized with large compilers.

In summary in this work:

- The rOEE python implementation has been improved as well as the parser to translate cQASM code into a graph.
- The rOEE algorithm has been coded in C++ and integrated into OpenQL that will be used for future experiments within the collaboration of the Technical University of Delft, Technical University of Catalonia and Technical University of Valencia.
- A Python script has been coded to calculate the non-local communications.
- The Cuccaro adder and the QFT adder have been implemented in Python.
- A parser to translate the QUEKO output to cQASM code has been developed.

4.2.3 Experimental framework

For executing the rOEE experiments, a simulation framework has been developed. For different experiments, different data comparisons will be performed. The framework will automatically recognize which comparisons and data are needed for each experiment by reading the directory name.

The framework supports the following experiments:

- Different range of physical or logical qubits.
- Different fixed numbers of physical or logical qubits
- Different densities of two-qubit gates.
- Different architectures: changing the number of cores, number of qubits per core, or changing both simultaneously.

It is also capable of providing the following data for comparison purposes:

- Number of qubits vs.: two-qubit gates, execution time, non-local communications, or the average of two-qubit gates.
- Number of slices vs.: no valid assignments or the average of two-qubit gates.
- Execution time vs.: non-local communications, two-qubit gates, the average of two-qubit gates, or the number of slices.
- Non-local communications vs. the average of two-qubit gates.
- Comparisons between different architectures: non-local communications and execution times.

The directory name will define the number of cores and qubits per core. The range of the logical qubits selected for performing the experiment will be given as the second parameter in the directory name. It can be a third parameter informing about other specific experiments; for example, the 'd' character will be written as the third parameter if we want to experiment with the density of two-qubit gates. The parameters will be separated by the ';' character.

For example, if the directory name is *10x10;50-100*, the framework will detect ten cores with ten physical qubits per core. The experiment will be run from 50 logical qubits to 100. A directory with the name *nxn;200* will be a weak scaling benchmarking, where the number of cores and qubits per core will vary, and it will have a fixed number of logical and physical qubits, 200.

The framework execution script will receive one argument, the directory path. It will have five optional arguments, all booleans, which refer to which framework parts want to be activated. These parts are:

- Make the graph. This option will execute the parser to translate the cQASM code of the quantum algorithm into a graph, taking into account the total physical qubits of the architecture. The data will be stored in a directory named *npz*. While doing this parse, other data, such as the number of total two-qubit gates the algorithm has, will be collected and stored in the *results/data* directory. To enable this option, the *graph* argument must be set to True.
- Execution of the rOEE algorithm. The rOEE algorithm will be executed, and the files from the *npz* directory will be read; the data will be saved in the directory *rOEE*. The execution time will be calculated and saved in the *time* directory. This option is enabled by setting the *rOEE* argument into True.
- Calculation of the results. The number of non-local communications, the times the algorithm fails, and slices will be calculated and saved in the *results/data* directory. To perform the results, the *results* argument must be equal to True.

- Generation of plots. Different data comparisons are made and saved in the *results/-figures* directory. It is necessary to set the *figures* argument into True to generate the graphics.
- Experiments comparison. This option compares data from different experiments and saves them in the main directory. The *comparison* argument must be set to True to activate this function.

5 Results

In this chapter, the experiments performed to analyze the performance of the rOEE algorithm and the results will be presented. We will perform the following experiments. First, we will try to replicate the results shown in the original work [10] to verify our implementation and make a comparison. Then, we will analyze how the rOEE algorithm performs when varying relevant quantum algorithm characteristics such as the gate density or when varying architectural parameters such as the number of qubits per core. Finally, we will perform a scalability analysis in which we will increase the number of qubits of the algorithm and perform a weak and strong scaling of the architecture.

For all sections of this chapter two metrics will be used: i) the non-local communications that corresponds to the qubits inter-core movements (SWAPs), and ii) the execution time that is the time the rOEE algorithm takes to find valid assignements for all time slices. In addition, all-to-all connectivity (inter-core and intra-core) and Exponential Decay for the lookahead weight will be used. Also, before starting the execution of the rOEE, the rOEE will be run for the first time slice, and the resulting assignment will be used as initial placement.

5.1 Comparison with original work

The first experiment will consist in comparing the results obtained with the rOEE code used in this work with the original one [10] for the Cuccaro and the QFT adder. Also, QUEKO and Quantum Volume benchmarks will be analyzed. In this case, a hundred physical qubits have been fixed for all experiments, and an architecture of ten cores and ten qubits per core will be use, i.e. 10x10.

Figure 14 shows the number of non-local communications versus the number of logical qubits that goes from 50 to 100 for all four benchmarks. Note that all of them follow the same trend: the number of non-local communications linearly increases with the number of logical qubits. The Cuccaro adder plot is similar to the graph shown in the original work, being the one with the least non-local communications. It has the fewest two-qubit gates and, after QUEKO, the smallest depth (see Table 4), so the total non-local communications of all slices are not expected to be very high.

The QFT adder shows the same trend as the original work, with an increase in the non-local communications because of the gate decomposition explained in section 4.1. It is the benchmark with the highest number of non-local communication. We can relate it to the high number of two-qubit gates involved in the quantum algorithm.

After the Cuccaro adder, the QUEKO benchmark has the least non-local communications. Again, it coincides with the number of two-qubit gates, having more than the Cuccaro adder but less than Quantum Volume. It should be noted that even having less depth than the Cuccaro adder, it has more non-local communication, which reaffirms the

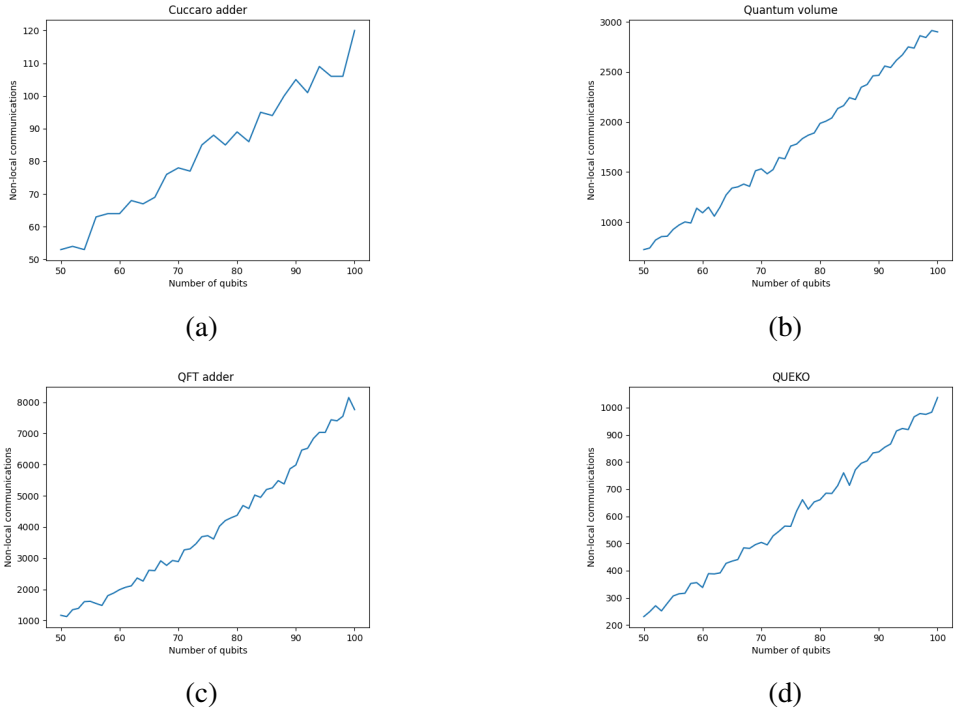


Figure 14. Number of non-local communications vs. number of logical qubits for (a) Cuccaro adder, (b) Quantum Volume, (c) QFT adder, and (d) QUEKO. An architecture of $10x10$ (cores x qubit) with 100 physical qubits is used.

fact that the number of two-qubit gates is the most relevant parameter for the number of non-local communications.

The amount of non-local communications for the Quantum Volume is between the values of QUEKO and the QFT adder since the number of two-qubit gates is in between both.

From the analyzed data, we can conclude that the most important parameters regarding the non-local communications (ordered from most relevant to the least one) are:

1. The higher the number of two-qubit gates, the higher the non-local communications.

2. The depth will impact the total non-local communication due to the sum of all non-local communication per slice but is not as important as the number of two-qubit gates.

3. The average of two-qubit gates may influence non-local communications due to increased constraints on finding a valid assignment. In addition, by choosing

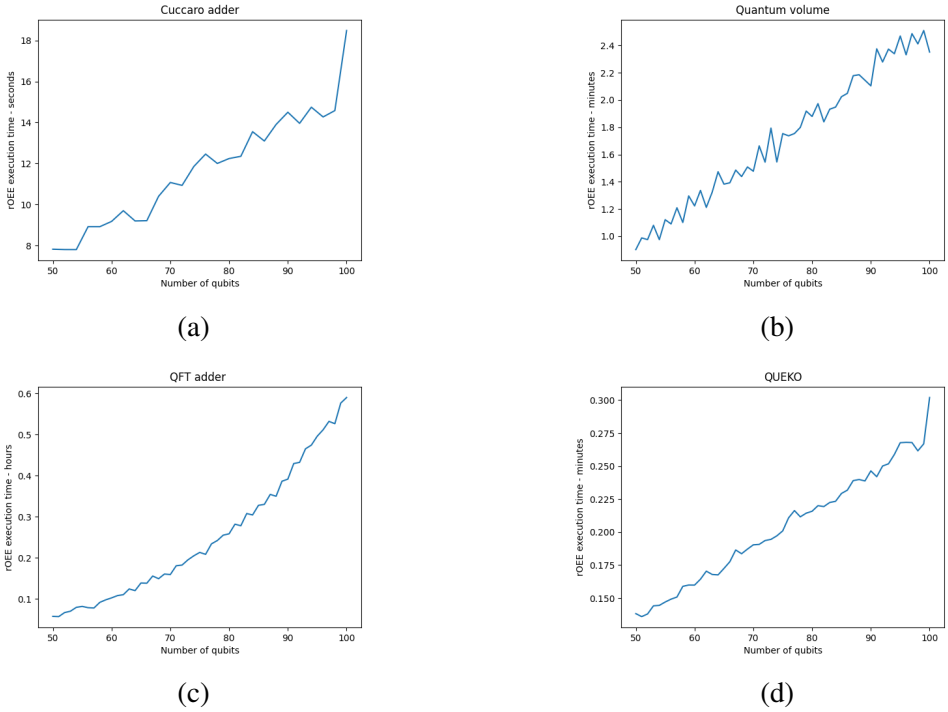


Figure 15. rOEE execution time vs. number of logical qubits for (a) Cuccaro adder, (b) Quantum Volume, (c) QFT adder, and (d) QUEKO. An architecture of $10x10$ (cores x qubit) with 100 physical qubits is used.

optimal lookahead weights in which qubits are exchanged considering future qubits interactions, the number of non-local communications can be reduced.

All plots show that the rOEE execution time increases linearly with the number of qubits (Figure 15), but also with the number of non-local communications, time slices, and the average of two-qubit gates (Figure 16). However, the QFT algorithm seems to increase exponentially with the number of qubits. One of the possible reasons is the high depth this benchmark has since, for each time slice, the rOEE algorithm must be executed, taking much time to go through all slices. We can also observe another difference in the QFT adder. In Figure 16 where the execution time and the average of two-qubit gates are compared, the time decreases instead of increasing; this is because the average of two-qubit gates decreases when the qubits increase, resulting in a sequential algorithm, as explained before.

The QFT adder presents the higher rOEE execution time, followed by the Quantum Volume, QUEKO, and Cuccaro adder, respectively. This order follows the non-local communications pattern we have seen before, being one of the reasons for how long the execution time is. In addition, other parameters such as depth, how parallel the algorithm

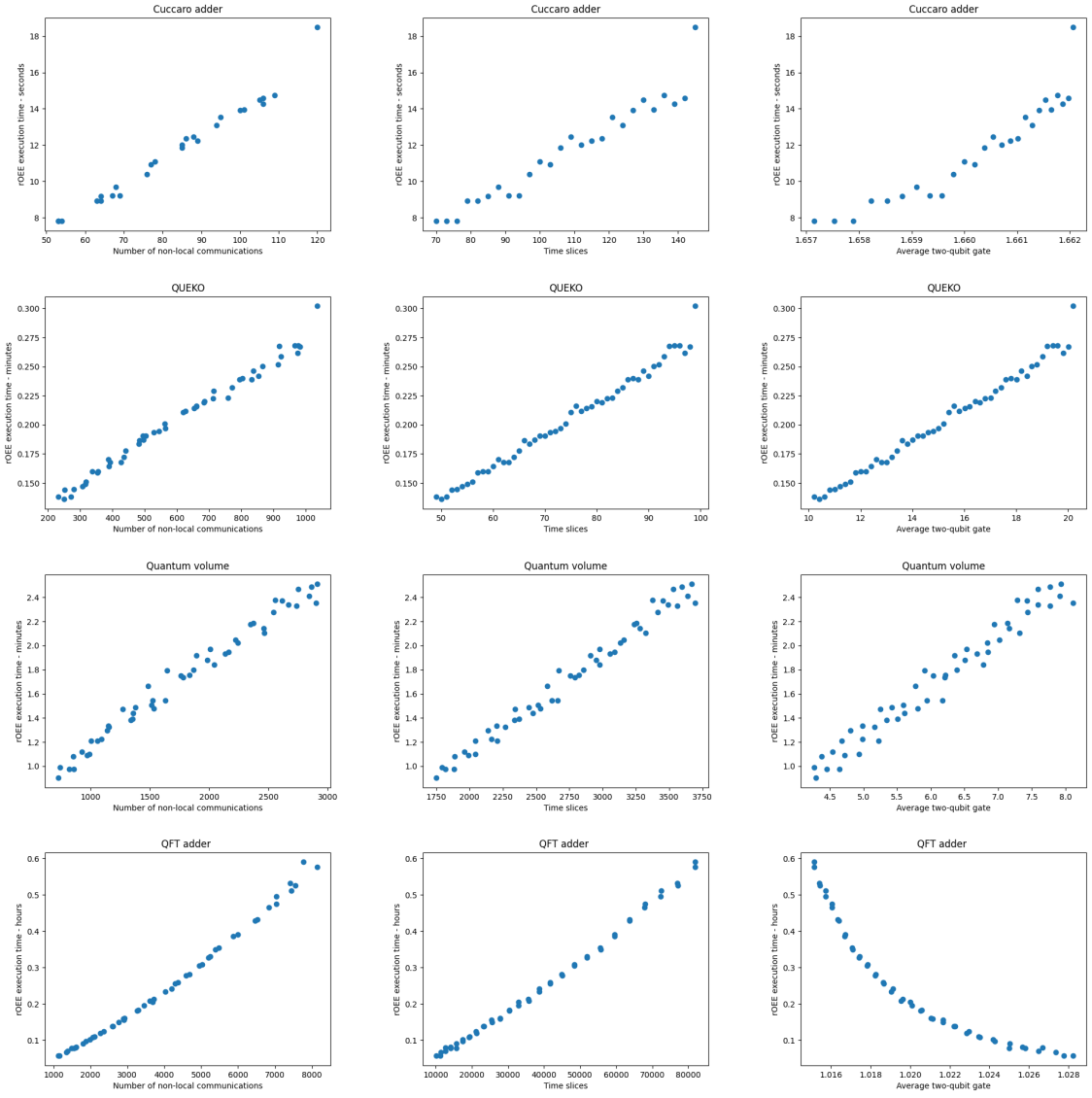


Figure 16. From the first to the last row: Cuccaro adder, QUEKO, Quantum Volume, and QFT adder. An architecture with 10 cores and 10 qubits per core ($10x10$) is assumed. The first column shows the time to complete the rOEE algorithm vs. the number of logical qubits involved in the algorithm. The rOEE execution time vs. the number of time slices is shown in the second column. In the last column, the time execution vs. the density of the two-qubit gates (x-axis) is shown.

is, or how many gates also affect how long the rOEE algorithm will take.

5.2 Varying the gate density of the quantum algorithm

The following experiment consists of changing the density gates in the QUEKO benchmark. The density is defined as $d = M/(n \cdot l)$, where M is the number of two-qubit gates, n the logical qubits, and l the longest dependency chain. The number of logical qubits will be fixed, varying the number of two-qubits gates and the longest dependency chain. This experiment will show how the rOEE algorithm behaves regarding different two-qubit gate densities for a fixed number of logical and physical qubits.

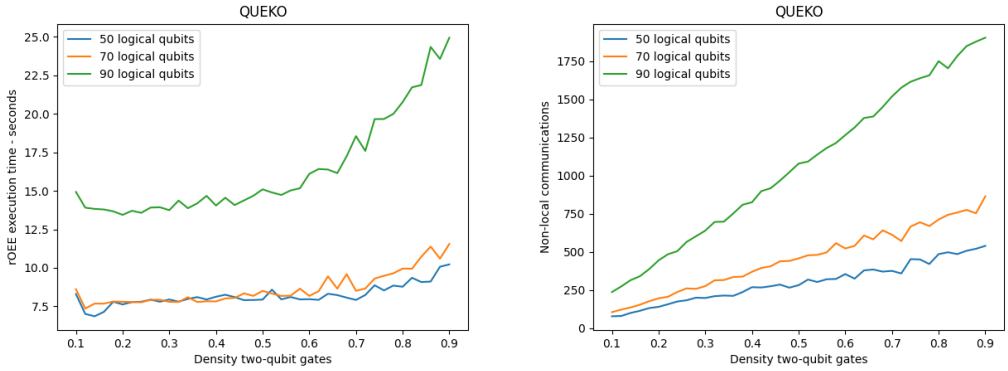


Figure 17. An architecture of 10 cores and 10 qubits per core ($10x10$) is assumed. The density of two-qubit gates changes from 0.1 to 0.9 by 0.02. Different logical qubits will be set: 90, 70, and 50. The time taken to complete the algorithm vs. the different two-qubit gates density is shown in the first plot. The second plot shows the non-local communications vs. the two-qubit gates' densities.

A 10x10 (cores x qubits) architecture with a hundred physical qubits is used. Three different numbers of logical qubits have been set: 50, 70, and 90. The density of two-qubit gates will go from 0.1 to 0.9. We will show how varying the gate density affects the rOEE execution time and non-local communications.

We can see the experiment results in Figure 17. First, as it has been shown in previous experiments, we can observe how the non-local communications are closely related to the execution time; that is, the higher the non-local communications, the higher the execution time (see green line corresponding to 90 logical qubits).

A straightforward observation is that the higher the two-qubit gate density is, the more non-local communications are needed. The more two-qubit gates per slice there are, the more constraints the qubits have to find a valid assignment for a specific time slice, and more SWAPs will be performed, as explained in the last section.

Another interesting result is that the 50 and 70 logical qubits are very similar in execution time and density of two-qubit gates, but we can observe a gap between them and 90 logical qubits. We already know that the two-qubit gate density makes a difference regarding non-local communications and time. However, the parameter that has more impact on the rOEE algorithm is the number of logical qubits used. When the number of logical qubits is closer to the physical qubits, there will be more constraints to find a valid assignment, as there will be fewer free qubits on the cores, which translates into more non-local communications and execution time.

5.3 Changing architectural parameters

In [10], a single multi-core architecture configuration consisting of 10 cores with 10 qubits per core (100 physical qubits) is used. In this section, we will analyze how the rOEE algorithm behaves when the number of qubits per core is changed. More specifically, we will keep the number of cores (10) but change the number of physical qubits per core depending on the number of logical qubits. We will show how the algorithm behaves in different architectures where the number of logical qubits is close to or equal to the number of physical qubits. As shown in the previous section, the number of non-local communications and the execution time increase considerably when the logical qubits are close to the physical qubits.

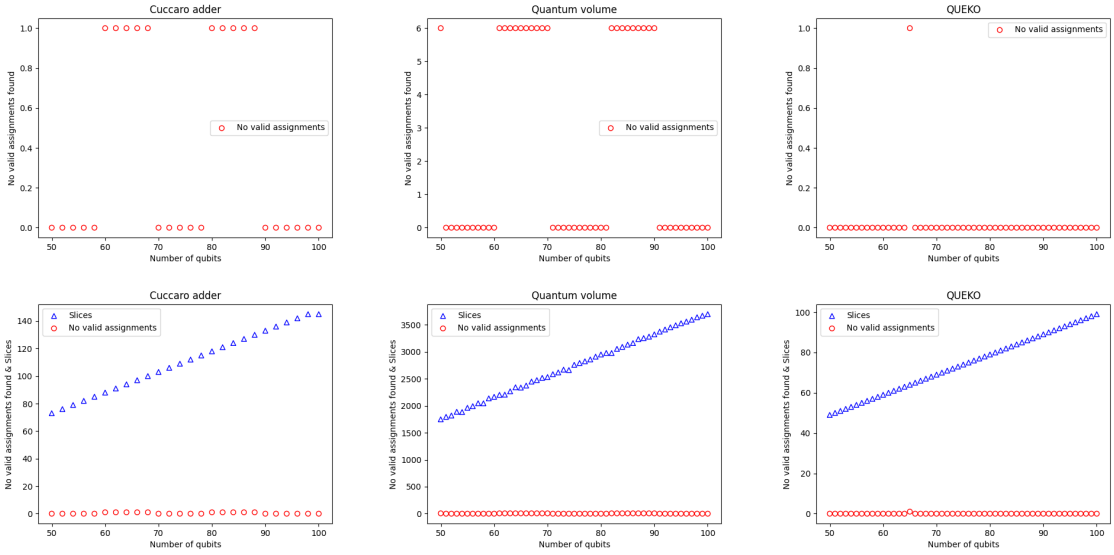


Figure 18. Number of non-valid assignments vs. the number of physical qubits for the Cuccaro adder, Quantum Volume, and QUEKO benchmarks. An architecture of $10xn$ (cores x qubit) is used, from 50 to 100 physical qubits.

As we just mentioned, in this experiment, the number of total cores is fixed to 10,

and the number of physical qubits per core varies depending on the current logical qubits. The number of physical qubits per core will be rounded up according to the number of logical qubits. For instance, if the algorithm has 61 logical qubits and the architecture consists of ten cores, $6.1 \simeq 7$ physical qubits per core will be needed; in total, 70 physical qubits.

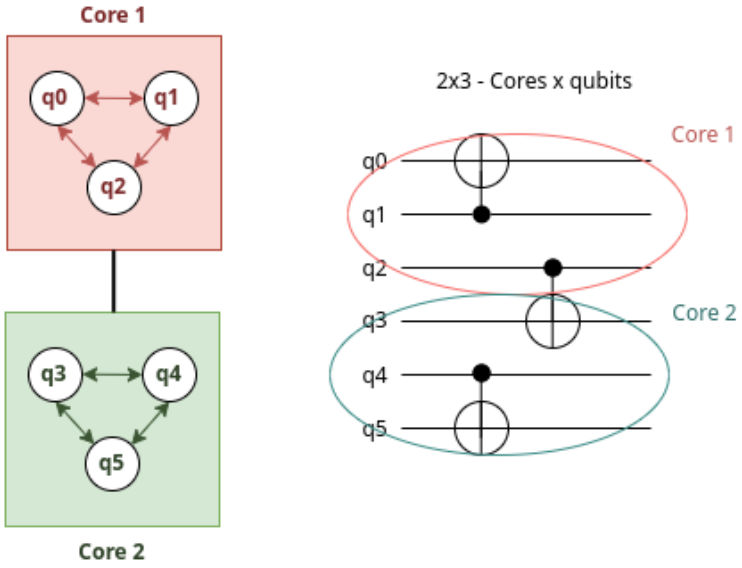


Figure 19. An example of a circuit for which the rOEE algorithm will not be able to find a valid assignment.

We observed that when an odd number of physical qubits per core is set, sometimes the rOEE algorithm is not able to find a valid assignment, as it is shown in Figure 18. To understand why rOEE fails, the problem can be reduced to a simple example as shown in Figure 19. Let's imagine a $2x3$ (cores x qubit) architecture with six physical qubits in which we want to run a circuit also consisting of 6 qubits (see Figure 19). The first three qubits of the quantum circuit are placed in core 1, and the other three to core 2. As shown in that time slice, all 3 gates can be performed in parallel, but a problem arises with qubits $q2$ and $q3$ since they must be on the same core to perform the gate, but this is not possible since all cores are full. The same happens in Cuccaro adder, QUEKO, and Quantum Volume on a larger scale.

As shown in the second row of Figure 18, the rOEE algorithm only fails one or a few times compared to the time slices, but one failure is enough to not be able to perform the mapping. To verify this, it can be seen in Figure 20 that the rOEE algorithm always works properly for an even number of physical qubits per core.

In summary, when varying the number of physical qubits depending on the algorithm

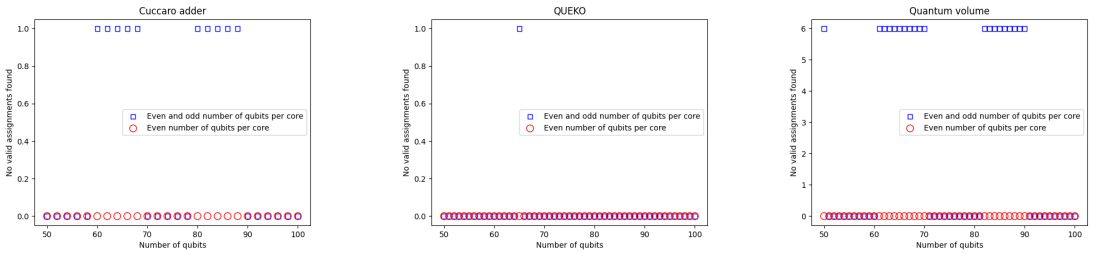


Figure 20. The number of non-valid assignments when an odd and even number of qubits per core is considered (as in Figure 19), or when only an even number of qubits per core is allowed. The number of cores is fixed to 10.

requirements (number of logical qubits) instead of fixing it to 100 (10x10), the rOEE algorithm might fail and not find a valid assignment when the number of qubits per core is odd.

The first column of Figure 21 shows the number of non-local communications for an even and odd, and only even, variable number of physical qubits per core. It should be noted that the data for the even and odd variable number of physical qubits per core is not entirely valid as we already know the algorithm sometimes fails, so figures are orientative. The Quantum Volume plot is the most meaningful since it is the one that fails the most when an odd number of qubits per core is set. Peaks can be observed in non-local communications when an odd number of physical qubits is fixed. These peaks disappear when only an even number of physical qubits is used, describing the difficulty of finding a valid assignment to satisfy the impossible constraints that we have seen in the example of Figure 19. Those peaks cannot be seen in other examples since the rOEE algorithm does not fail as many times, so the rOEE algorithm is not that stressed. The more difficult it is to find a valid assignment, the more non-local communications we will have.

A comparison between non-local communication for an even variable and a fixed number of physical qubits is shown in the second column. Again, it is assumed an architecture of ten cores. The same behavior can be observed in the four graphics; the non-local communications are almost the same for a fixed and an even variable number of qubits, being the variable number of qubits slightly higher, telling us that the number of physical qubits in the architecture is not an essential factor to take into account regarding the non-local communications.

In contrast, the total number of physical qubits is crucial for the rOEE runtime, as shown in the third column. When the number of physical qubits is fixed, and the logical qubits increase, the execution time scales linearly, but the time scales considerably when the physical qubits increase. The rOEE algorithm computes over physical qubits to find valid assignment, so the more physical qubits, the more operations it performs, increasing the execution time. This experiment shows that the number of physical qubits

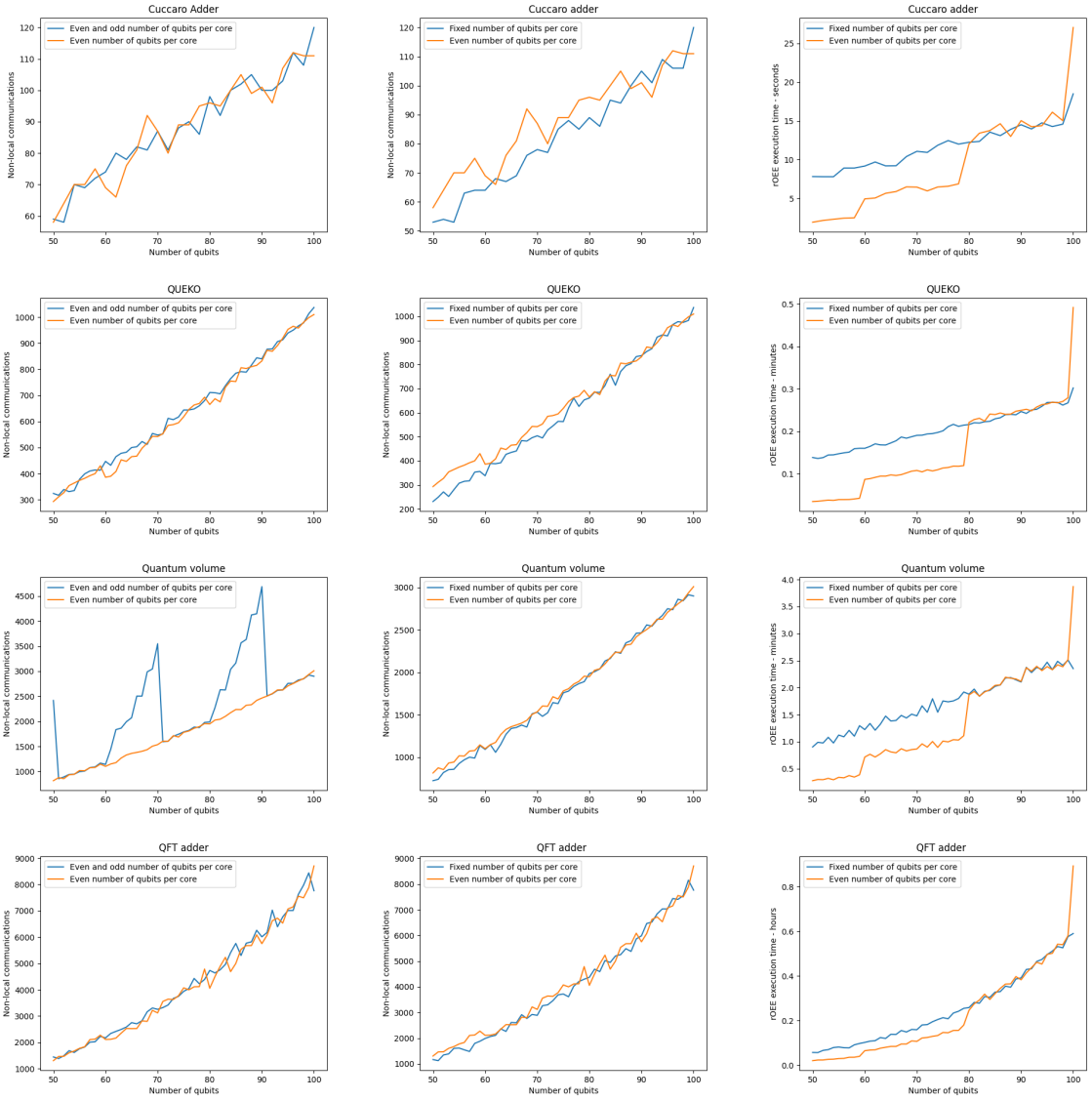


Figure 21. From the first to the last row: Cuccaro adder, QUEKO, Quantum Volume, and QFT adder. An architecture with ten fixed cores is assumed. In the first column, the number of non-local communications vs. the number of logical qubits is shown, the number of physical qubits varies depending on the number of logical qubits, with one only allowing an even number of qubits per core. Non-local communications (second column) and execution time (third column) for an architecture with the number of physical qubits fixed and an even variable number of physical qubits per core is shown. This data is compared with the number of logical qubits used.

an architecture has can have a limit due to execution time.

5.4 Analyzing the scalability of the rOEE algorithm

In this section, we will explore how the rOEE algorithm behaves when the number of qubits of the algorithm increases up to 500 qubits for the Cuccaro adder, 250 qubits for the Quantum Volume, and QUEKO, and 160 for the QFT adder. We will consider two architectures: one in which the number of physical qubits is fixed to the maximum number of logical qubits for each algorithm (i.e. 500 physical qubits for the Cuccaro adder, 250 qubits for the Quantum Volume and QUEKOand 160 for the QFT adder), and another in which the number of physical qubits will vary with even numbers of qubits per core. The maximum number of logical qubits in each benchmark has been chosen depending on how long the rOEE algorithm takes to execute.

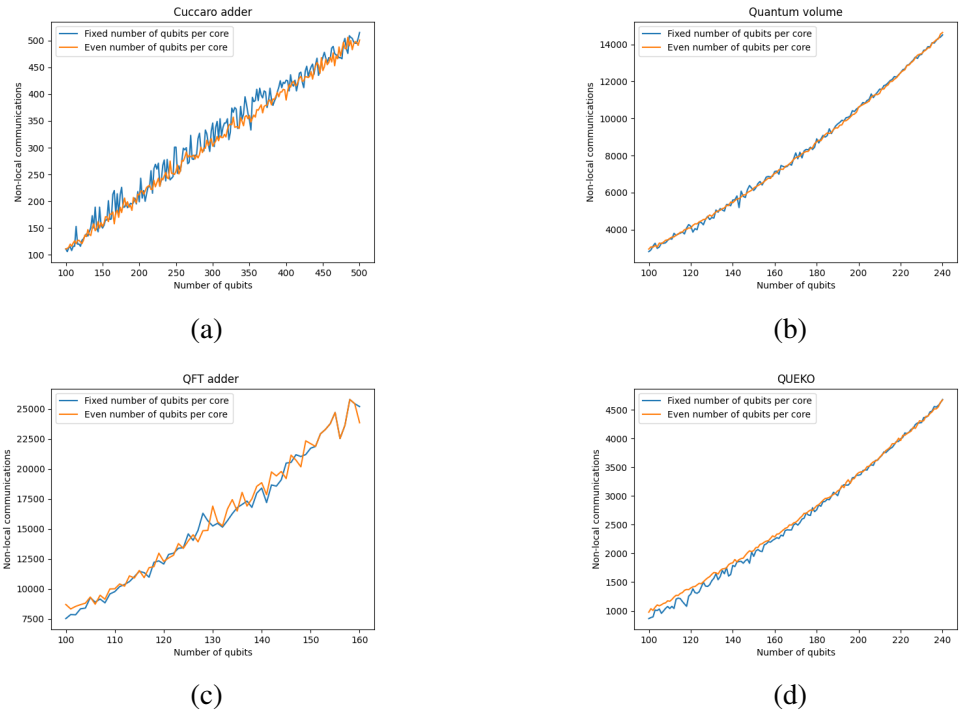


Figure 22. Non-local communications vs. number of logical qubits for (a) Cuccaro adder, (b) Quantum Volume, (c) QFT adder, and (d) QUEKO. An architecture with the number of physical qubits fixed and an even variable number of physical qubits per core is assumed.

In Figure 22 and Figure 23 the results of the scalability experiment are shown. The same non-local communication and execution time behavior is observed for all benchmarks as in the previous section. Since the Cuccaro adder is the algorithm with

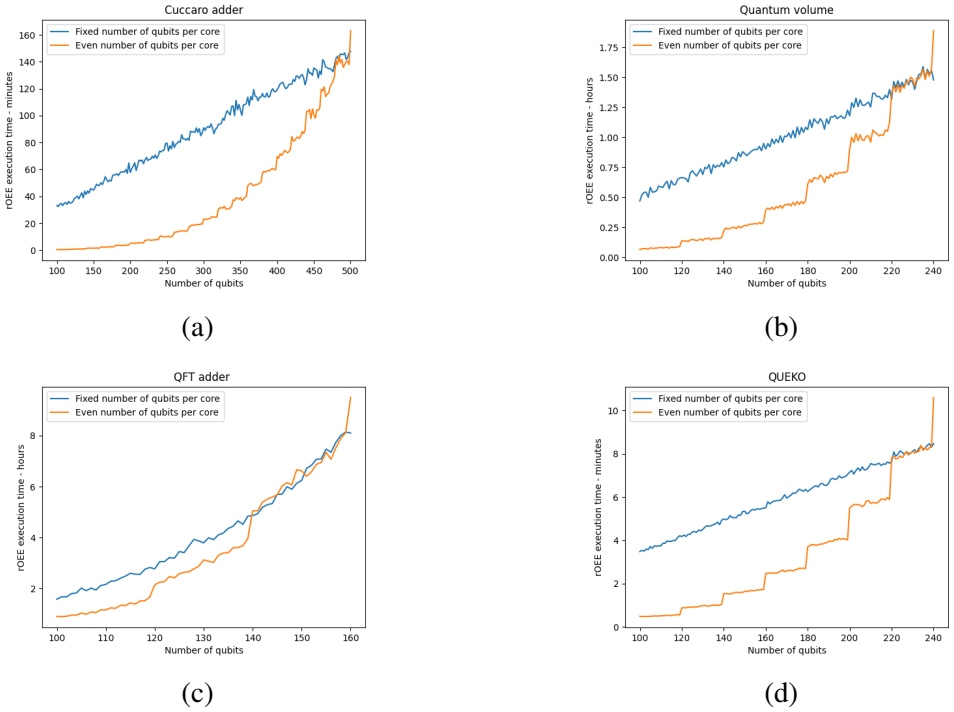


Figure 23. rOEE execution time vs. number of logical qubits for (a) Cuccaro adder, (b) Quantum Volume, (c) QFT adder, and (d) QUEKO. An architecture with the number of physical qubits fixed and an even variable number of physical qubits per core is assumed.

the shortest execution time, it can be scaled up to 500 physical qubits. After that, it becomes unfeasible in terms of execution time. Quantum Volume and QUEKO will run until 240 physical qubits are reached. The QFT adder is the algorithm with the highest depth and number of two-qubit gates, so it has the longest execution time; we were only able to scale it up to 160 physical qubits since the runtime is too long (more than 8 hours per execution). The rOEE algorithm could be implemented in another programming language such as C or Julia to improve the execution time, so the maximum number of physical qubits could be increased.

The execution time scales linearly when physical qubits are fixed and exponentially when the number of physical qubits is variable. As in the previous section, we can appreciate that the execution time increases considerably when the physical qubits increase, being more optimal to execute the rOEE algorithm with fewer physical qubits.

5.5 Analyzing the scalability of the multi-core architecture

As defined for classical computing systems, "scalability is the property of a system to handle a growing amount of work by adding resources to the system" [46]. Following

this definition of scalability, we have performed two types of scalability experiments:

- Weak scaling: the total number of physical qubits is fixed, and the number of cores and qubits per core vary. For example, for 100 physical and logical qubits, the architecture will scale as follows: first, it will consist of 2 cores of 50 qubits each, then 4 cores of 25 qubits each, up to 50 cores of 2 qubits each.

- Strong scaling: the number of qubits per core is fixed, and the total qubits increase when adding more cores. For example, if 10 qubits per core are fixed, the architecture will have 2 cores and 20 total qubits, 3 cores and 30 total qubits, and up to 10 cores with 100 total qubits.

When performing the weak scaling, 200 physical and logical qubits have been considered, except for the QFT adder, which will be 100 because of the excessive execution time. The number of cores will increase, and the number of qubits per core will decrease until the maximum number of cores for the physical qubits is reached. Note that the number of qubits per core must be even since the rOEE algorithm may fail with an odd number of qubits per core.

In Figure 24 the weak scaling results are shown. The rOEE runtime and non-local communications increase as more cores are involved in the execution. With more cores, more inter-core movements will be until rOEE finds a valid assignment, and more constraints will be since fewer qubits will be available per core, which translates into more execution time. The non-local communications scale is similar for all benchmarks; meanwhile, the execution time differs. The runtime rises more when the number of cores is increased than with the non-local communications.

In the Cuccaro adder, the execution time increases drastically when the number of cores is increased, the same happens with the Quantum Volume benchmark. QUEKO, on the other hand, the runtime increases slower than other benchmarks.

Strong scaling will increase the number of cores and total physical qubits while the number of qubits per core is fixed. Four different experiments have been done; they all start with 2 cores and end with 100 cores: with 4, 6, 8, and 10 qubits per core fixed. In Figure 25 the strong scaling is shown with 4,6,8 and 10 qubits per core.

All graphics comparing the non-local communication and number of qubits used show an exponential behavior except the Cuccaro adder since it this the one with less depth and two-qubits gates. The exponential trend increases when using fewer qubits per core since more inter-core movements will be as more constraints are to find a valid assignment.

The execution time regarding the total number of qubits used is practically the same for 4,6,8 and 10 qubits per core, since the most crucial parameter concerning execution time in the rOEE algorithm is the total number of physical qubits used, as shown in previous sections. When comparing the execution time with the non-local communications, the data between different numbers of qubits per core differ. This is

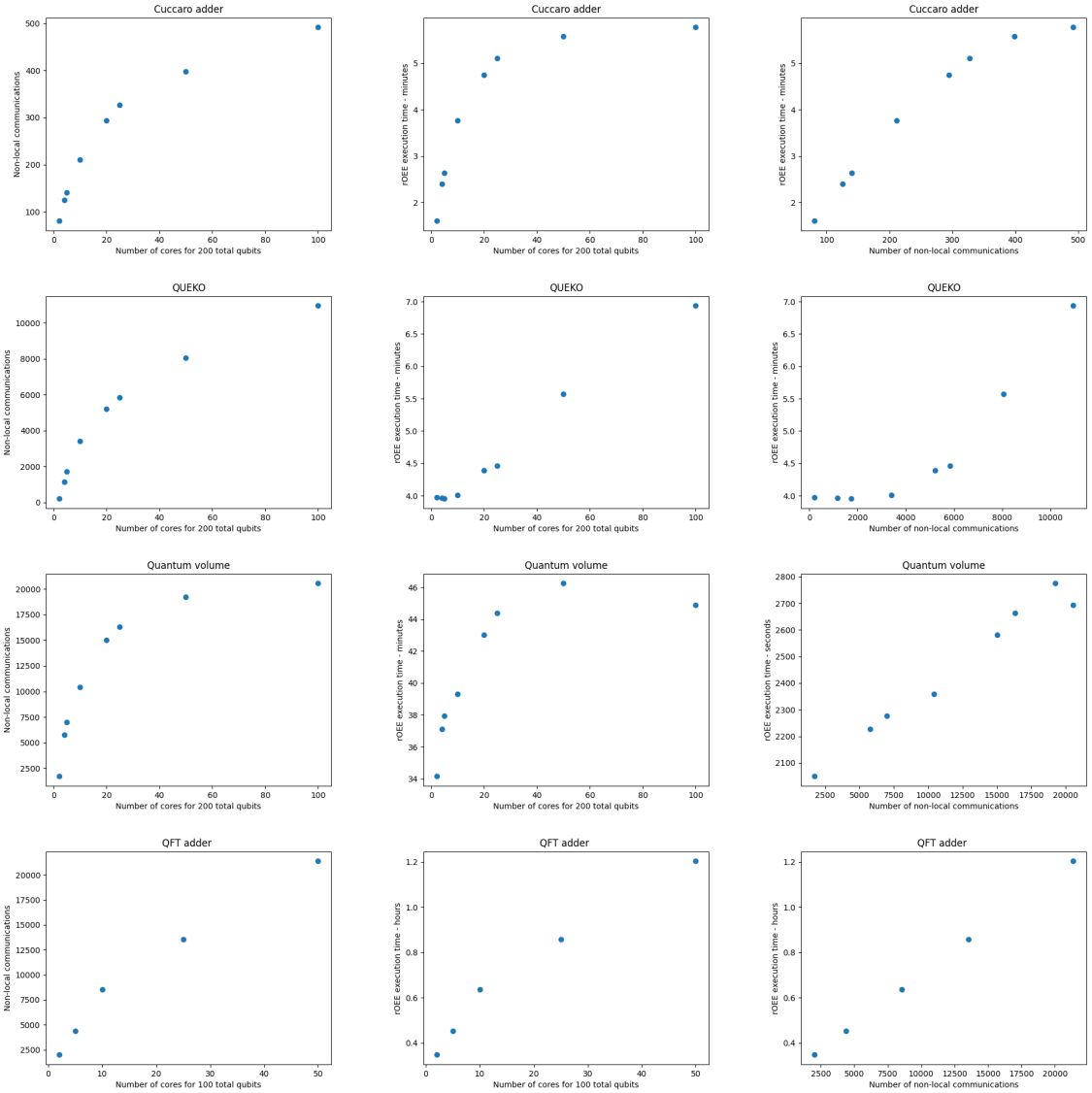


Figure 24. Weak scaling. From the first to the last: Cuccaro adder, QUEKO, Quantum Volume, and QFT adder. Physical and logical qubits are set to 200, except in the QFT adder, which is set to 100. The number of cores increase, and the number of even qubits per core will decrease until the maximum number of cores for the physical qubits is reached. The first column compares non-local communications with the number of cores. The algorithm runtime is compared with the number of cores (second column) and non-local communications (third column).

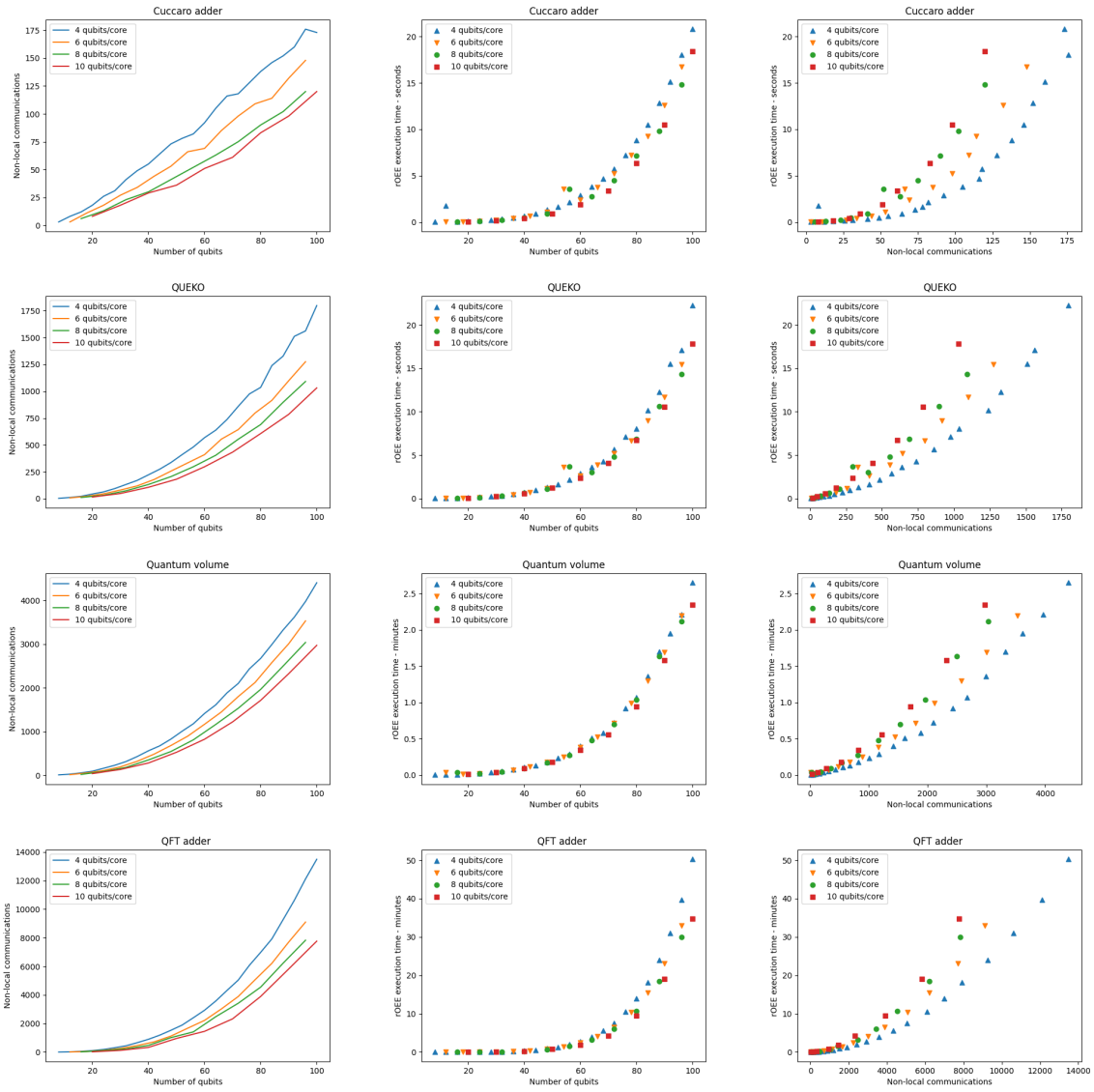


Figure 25. Strong scaling. From the first to the last: Cuccaro adder, QUEKO, Quantum Volume, and QFT adder. Physical qubits will vary from 8 to 100 qubits, and four different architectures will be compared: 4,6,8, and 10 qubits per core; logical qubits will increase according to physical qubits. The number of cores will increase with the total number of physical qubits. The first column compares non-local communications with the number of physical qubits. In the second and third columns, each point refers to the number of cores used. The algorithm runtime is compared with the number of physical qubits used (second column) and non-local communications (third column).

because, in order for the number of non-local communications corresponding to different numbers of qubits per core to be the same, the total number of cores involved is not equal, having a different total of physical qubits.

6 Conclusions and future work

Multi-core quantum computing architectures in which several quantum processors are connected via coherent links and classical communication channels are one of the most promising architectural proposals for scaling up quantum systems. New challenges arise with this novel approach, such as developing quantum circuit mapping techniques for quantum algorithms to be successfully run on such architectures. So far, only one mapping algorithm for multi-core quantum processors has been proposed, which is based on the relaxed Overall Extreme Exchange (rOEE) [10]. However, this approach has not been studied in depth.

In this thesis, the rOEE algorithm has been further analyzed for its application in mapping quantum algorithms on multi-core quantum architectures. For this purpose, four different quantum algorithms have been used as benchmarks: Cuccaro adder, QFT adder, Quantum Volume, and QUEKO. In addition, several experiments have been performed in which architectural as well as algorithm parameters have been changed:

- First, an architecture of ten cores with ten qubits per core has been considered, increasing the logical qubits from 50 to 100 as done in [10]. The results show a linear trend in non-local communications and rOEE execution time for all algorithms, except for the QFT adder, which has an exponential trend in execution time.

- Secondly, we have modified the density of the two-qubit gates for the QUEKO benchmarks, which results in more non-local communications and execution time as the density of the two-qubit gates increases. Furthermore, it can be shown that a crucial parameter when executing the rOEE algorithm is the number of logical qubits used regarding the total physical qubits.

- Thirdly, we have set the number of qubits per core based on the logical qubits used in the quantum algorithm, with an architecture of ten cores. In this case, we observed that the rOEE algorithm might fail when an odd number of physical qubits is used. Also, the rOEE execution time drastically increases when the number of physical qubits increases.

- Then, we have scaled the number of the physical qubits of the architecture with two different architectures: a fixed number of qubits per core and varying the number of qubits per core based on the logical qubits. The results show that using a variable number of qubits per core is more efficient in terms of execution time.

- Finally, a weak and strong scaling of the multi-core architecture has been performed. For the weak scaling, we can observe that the rOEE execution time and non-local communications increase as more cores are involved in the execution.

The strong scaling shows an exponential trend in non-local communications; also, it shows that the most crucial parameter in terms of execution time is the total number of physical qubits used.

After performing the experiments, it has been shown that algorithms with a large depth, a high number of two-qubit gates (such as the QFT adder), and architectures with a considerable amount of physical qubits and cores can take a long time to find the result due to the number of iterations the rOEE algorithm must perform. In addition, the rOEE algorithm fails when an odd number of qubits per core is set because of the restrictions on finding a valid assignment. Also, when the number of logical qubits approaches the number of physical qubits, there will be more execution time and non-local communications as there will be more constraints to find a valid assignment.

Also, some contributions have been made while working on the thesis:

- The Cuccaro adder and the QFT adder have been coded in Python.
- The rOEE algorithm has been implemented in C++ and integrated into OpenQL.
- The rOEE python code has been improved, some bugs were solved, and a script has been implemented to calculate non-local communications.
- A parser to translate the QUEKO output to cQASM has been coded.
- A framework has been created to automatically run the rOEE algorithm experiments and save the results.

Based on the results of this thesis, future work can be done to improve the rOEE algorithm. When an odd number of qubits per core is set, and there is a failure, the two-qubit gate that is impossible to perform can be postponed to a new time slice. In addition, for architectures with a considerable amount of physical qubits that might take too long to execute when performing the rOEE algorithm, the physical qubits can be set based on the logical qubits if there is a considerable difference between them, making some physical qubits unusable to reduce the execution time. For example, if the architecture has 100 physical qubits, but only 50 have been used, the rOEE could work with 70 physical qubits. The last proposed improvement is based on the lookahead weight, optimizing the function it uses to achieve better execution times and fewer non-local communications.

Bibliography

[1] Andreas Trabesinger. "Quantum simulation". In: *Nature Physics* (2012). DOI: https://doi.org/10.1038/nphys2258.

[2] William Coffeen Holton. In: (2021). URL: https://www.britannica.com/technology/quantum-computer#ref829410.

[3] Nicholas LaRacuente et al. "Short-Range Microwave Networks to Scale Superconducting Quantum Computation". In: (2022). arXiv: 2201.08825 [quant-ph].

[4] C Monroe et al. "Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects". In: *Physical Review A* 89.2 (2014), p. 022317.

[5] Jay Gambetta. "Expanding the IBM Quantum roadmap to anticipate the future of quantum-centric supercomputing". In: (2021). URL: https://research.ibm.com/blog/ibm-quantum-roadmap-2025.

[6] John Preskill. "Quantum Computing in the NISQ era and beyond". In: *Quantum* 2 (Aug. 2018), p. 79. ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79. URL: http://dx.doi.org/10.22331/q-2018-08-06-79.

[7] Marcos Yukio Siraichi et al. "Qubit Allocation". In: CGO 2018 (2018), pp. 113–125. DOI: 10.1145/3168822. URL: https://doi.org/10.1145/3168822.

[8] Lingling Lao et al. "Mapping of quantum circuits onto NISQ superconducting processors". In: (Aug. 2019).

[9] Alwin Zulehner, Alexandru Paler, and Robert Wille. "An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.7 (2019), pp. 1226–1236. DOI: 10.1109/TCAD.2018.2846658.

[10] Jonathan M. Baker et al. "Time-sliced quantum circuit partitioning for modular architectures". In: *Proceedings of the 17th ACM International Conference on Computing Frontiers* (May 2020). DOI: 10.1145/3387902.3392617. URL: http://dx.doi.org/10.1145/3387902.3392617.

[11] Azure Microsoft. "Overview what is quantum computing". In: (2020). URL: https://azure.microsoft.com/en-us/overview/what-is-quantum-computing/%5C#real-world-uses.

[12] Michael A. Nielsen and Isaac L. Chuang. "Quantum Computation and Quantum Information". In: (2000).

[13] IBM. In: (2016). URL: https://quantum-computing.ibm.com/composer.

[14] Fei Yan et al. "Tunable Coupling Scheme for Implementing High-Fidelity Two-Qubit Gates". In: *Physical Review Applied* 10 (Mar. 2018). DOI: 10.1103/PhysRevApplied.10.054062.

[15] "Eagle IBM Quantum Processor". In: (2021). URL: https://research.ibm.com/blog/127-qubit-quantum-processor-eagle.

[16] C Ospelkaus et al. "Trapped-Ion Quantum Logic Gates Based on Oscillating Magnetic Fields". In: *Physical review letters* 101 (Sept. 2008), p. 090502. DOI: 10.1103/PhysRevLett.101.090502.

[17] "IonQ Aria Furthers Lead As World's Most Powerful Quantum Computer." In: (2022). URL: https://ionq.com/news/february-23-2022-ionq-aria-furthers-lead.

[18] Maximilian Ruf et al. "Quantum networks based on color centers in diamond". In: *Journal of Applied Physics* 130.7 (Aug. 2021), p. 070901. ISSN: 1089-7550. DOI: 10.1063/5.0056534. URL: http://dx.doi.org/10.1063/5.0056534.

[19] Tong Wu and Jing Guo. "Variability and Fidelity Limits of Silicon Quantum Gates Due to Random Interface Charge Traps". In: *IEEE Electron Device Letters* 41.7 (2020), pp. 1078–1081. DOI: 10.1109/LED.2020.2997009.

[20] Dolev Bluvstein et al. "A quantum processor based on coherent transport of entangled atom arrays". In: *Nature* 604.7906 (Apr. 2022), pp. 451–456. DOI: 10.1038/s41586-022-04592-6. URL: https://doi.org/10.1038%2Fs41586-022-04592-6.

[21] C. J. Picken et al. "Entanglement of neutral-atom qubits with long ground-Rydberg coherence times". In: (2018). DOI: 10.48550/ARXIV.1808.04755. URL: https://arxiv.org/abs/1808.04755.

[22] Quantum Computing state of the art. In: (2018). URL: https://www.bcg.com/publications/2018/next-decade-quantum-computing-how-play.

[23] "Algorithmic Qubits: A Better Single-Number Metric". In: (2022). URL: https://ionq.com/posts/february-23-2022-algorithmic-qubits.

[24] "The IBM Quantum heavy hex lattice". In: (2021). URL: https://research.ibm.com/blog/heavy-hex-lattice.

[25] "QuEra Computing". In: (2019). URL: https://www.quera.com/.

[26] Andrew Wack et al. "Quality, Speed, and Scale: three key attributes to measure the performance of near-term quantum computers". In: (2021). DOI: 10.48550/ARXIV.2110.14108. URL: https://arxiv.org/abs/2110.14108.

[27] Yongshan Ding et al. "Systematic Crosstalk Mitigation for Superconducting Qubits via Frequency-Aware Compilation". In: (Oct. 2020), pp. 201–214. DOI: 10.1109/MICRO50266.2020.00028.

[28] ChowJM. "Quantum intranet". In: *IETQuant.Comm* 2.1 (2020). URL: https://doi.org/10.1049/qtc2.12002.

[29] Daniele Cuomo et al. “Optimized compiler for Distributed Quantum Computing”. In: (2021). arXiv: 2112.14139 [quant-ph].

[30] Nemanja Isailovic et al. In: (2006), pp. 366–377.

[31] Santiago Rodrigo et al. “Exploring a Double Full-Stack Communications-Enabled Architecture for Multi-Core Quantum Computers”. In: (2020). arXiv: 2009.08186 [quant-ph].

[32] Marcos Yukio Siraichi et al. “Qubit Allocation”. In: CGO 2018 (2018), pp. 113–125. DOI: 10.1145/3168822. URL: https://doi.org/10.1145/3168822.

[33] Alexander Cowtan et al. “On the qubit routing problem”. In: Leibniz International Proceedings in Informatics (LIPIcs) 135 (May 2019). Ed. by Wim van Dam and Laura Mancinska, 5:1–5:32. DOI: 10.4230/LIPIcs.TQC.2019.5. URL: https://doi.org/10.4230/LIPIcs.TQC.2019.5.

[34] Aaron Lye, Robert Wille, and Rolf Drechsler. “Determining the minimal number of swap gates for multi-dimensional nearest neighbor quantum circuits”. In: *The 20th Asia and South Pacific Design Automation Conference* (2015), pp. 178–183.

[35] Robert Wille et al. “Using pDDs for Nearest Neighbor Optimization of Quantum Circuits”. In: (2016). Ed. by Simon Devitt and Ivan Lanese, pp. 181–196.

[36] Davide Venturelli et al. “Compiling quantum circuits to realistic hardware architectures using temporal planners”. In: *Quantum Science and Technology* 3.2 (Feb. 2018), p. 025004. DOI: 10.1088/2058-9565/aaa331. URL: https://doi.org/10.1088%2F2058-9565%2Faaa331.

[37] Charles M. Fiduccia and Robert M. Mattheyses. “A Linear-Time Heuristic for Improving Network Partitions”. In: *19th Design Automation Conference* (1982), pp. 175–181.

[38] B. W. Kernighan and S. Lin. “An Efficient Heuristic Procedure for Partitioning Graphs”. English (US). In: *ATamp;T Technical Journal* 49 (Feb. 1970), pp. 291–307. ISSN: 8756-2324. DOI: 10.1002/j.1538-7305.1970.tb01770.x.

[39] George Karypis and Vipin Kumar. “METIS – Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0”. In: (1995).

[40] Taehoon Park and Chae Y. Lee. “Algorithms for partitioning a graph”. In: *Computers & Industrial Engineering* 28.4 (1995), pp. 899–909. ISSN: 0360-8352. DOI: https://doi.org/10.1016/0360-8352(95)00003-J. URL: https://www.sciencedirect.com/science/article/pii/036083529500003J.

[41] “OpenQL compiler”. In: (2016). URL: https://openql.readthedocs.io/en/latest/.

[42] Steven A. Cuccaro et al. “A new quantum ripple-carry addition circuit”. In: (2004). DOI: 10.48550/ARXIV.QUANT-PH/0410184. URL: https://arxiv.org/abs/quant-ph/0410184.

[43] Andrew W. Cross et al. “Validating quantum computers using randomized model circuits”. In: *Phys. Rev. A* 100 (3 Sept. 2019), p. 032328. DOI: 10.1103/PhysRevA.100.032328. URL: https://link.aps.org/doi/10.1103/PhysRevA.100.032328.

[44] Thomas G. Draper. “Addition on a Quantum Computer”. In: (2000). DOI: 10.48550/ARXIV.QUANT-PH/0008033. URL: https://arxiv.org/abs/quant-ph/0008033.

[45] Bochen Tan and Jason Cong. “Optimality Study of Existing Quantum Computing Layout Synthesis Tools”. In: *IEEE Transactions on Computers* 70.9 (2021), pp. 1363–1373. DOI: 10.1109/TC.2020.3009140.

[46] André B. Bondi. “Characteristics of Scalability and Their Impact on Performance”. In: WOSP ’00 (2000), pp. 195–203. DOI: 10.1145/350391.350432. URL: https://doi.org/10.1145/350391.350432.

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Anabel Ovide González**,
(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Multi-core Quantum Computers: Analyzing the mapping of quantum algorithms,
   (title of thesis)

   supervised by Dr. Carmen G. Almudever and Dr. Dirk Oliver Theis.
   (supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Anabel Ovide González
17/05/2022