Assessing the mapping of quantum algorithms on superconducting quantum processors

Daniel Moreno Manzano

UPC and TU Delft

Assessing the mapping of quantum algorithms

on superconducting
quantum processors

by

Daniel Moreno Manzano

in partial fulfillment of the requirements for the degree of

Master of Telecommunication Engineering

at the UPC,

Affiliation:	Quantum and Computer Architecture Lab Department of Quantum and Computer Engineering Faculty of Electrical Engineering, Mathematics and Computer Sciences Delft University of Technology	
Project duration:	February 5, 2018 – February 8, 2019	
Thesis committee:	Prof. dr. ir. José Antonio Rubio Sola,	Eng.Electrònica, UPC (Supervisor)
	Prof. dr. ir. Josep Altet Sanahujes,	Eng.Electrònica, UPC
	Prof. dr. ir. Adolfo Comerón Tejero,	Teoria del Senyal i Communications, UPC
Supervisors:	Prof. dr. ir. José Antonio Rubio Sola	Dr. ir. Carmina García Almudéver

Abstract

Quantum computers hold the promise for solving efficiently important problems in computational sciences that are intractable nowadays by exploiting quantum phenomena such are superposition and entanglement. Research in quantum computing is mainly driven by the development of quantum devices and quantum algorithms. Quantum algorithms can be described by quantum circuits, which are hardware agnostic – e.g it is assumed that any arbitrary interaction between qubits is possible. However, real quantum processors have a series of constraints that must be complied to when running a quantum algorithm. Therefore, a mapping process that adapts the quantum circuit to chip's constraints is required.

The mapping process will, in general, increase the number of gates and/or the circuit depth. As qubits and gates are error prone, it will result in an increment of the failure rate of computation while running the adapted quantum algorithm in a given quantum device. Most of the current mapping models optimize and are assessed based on two metrics: circuit depth (or latency) and number of (movement) operations added; they should be as minimal as possible. However, these metrics are not giving any information about how the mapping process is affecting the reliability of the algorithm. In other words, can still the algorithm produce 'good' results after being mapped?

The aim of this thesis is to propose some new mapping metrics that allow to study the impact of the mapping process on the algorithm's reliability. These are, quantum fidelity, probability of success of the algorithm and quantum volume. They could be used not only to assess the quality of the mapping procedure but also as parameters to be optimized by the mapping.

To this purpose, different quantum algorithms have been mapped into the superconducting quantum processor, called Surface-17, developed at QuTech.

Acknowledgements

I am very thankful to have had the opportunity of carried out this thesis. This research was made possible by the great support of my supervisors and mentors Carmina G. Almudéver, Lingling Lao and Prof. Koen Bertels. They accepted me besides my poor background on quantum computing and they were patient enough to teach me about it and about research. I also would like to separately thank my supervisor in Barcelona, Antonio Rubio, who offered me and facilitated the opportunity of developing this study.

I appreciated all the help offered by my lab colleagues Hans van Someren, Imran Ashraf, Andreas Papageorgiou, Leon Riesebos and Aritra Sarkar. We had interesting discussions that helped me to elucidate the solutions of the problems that appeared through out the research. I also would like to thank Savvas Varsamopoulos, Abid Moueddene, Prof. Koen, Vieri Mattei, Miguel Serrão, Aritra, Leon and Andreas for all the fun and the beers we had together.

In general, I would like to express my gratitude to all the QCE department, specially to Erik de Vries – who had deal with all my demands related with the lab servers –, Joyce van Velzen and Lidwina Tromp – who arranged all the burocracy requirements in TU Delft for me.

Thanks for all your support!

–

Agradezco mucho la compañía de mi familia en este tiempo, quienes no han parado de cuidar de mí. Sin ellos mi estancia en Los Países bajos hubiera sido inaguantable. Doy gracias a mi madre y a mi hermana por todas esas videollamadas que me han hecho sentir tan cerca de casa.

Finalmente, quiero agradecer la compañía de todos mis amigos en éste tiempo. Gracias por venir a visitarme y apoyarme también en la distancia. Especialmente a Carlos Robisco, Cristina Puigjaner y Germán Fañanás, quienes me han ayudado mucho a aclararme con mi vocación.

¡Muchas gracias a todos por haberme acompañado en éste tiempo!

Daniel Moreno Manzano
Delft, The Netherlands
February 2019

List of Acronyms

ALAP	As late as possible
ALU	Arithmetic logic unit
ASAP	As soon as possible
GNFS	General number field sieve
GPU	Graphics processing units
NN	Nearest neighbor
QEC	Quantum error correction
SC	SuperConducting

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Problem Statement . . . 1
- 1.3 Structure of the thesis . . . 2

2 Quantum computing and mapping of quantum circuits **3**
- 2.1 Quantum computing . . . 3
  - 2.1.1 Essential elements of quantum computation . . . 3
  - 2.1.2 Quantum Circuits . . . 5
  - 2.1.3 Qubits are faulty . . . 6
- 2.2 Mapping of quantum circuits . . . 6
  - 2.2.1 Initial placement . . . 6
  - 2.2.2 Routing . . . 6
  - 2.2.3 Scheduling . . . 7
  - 2.2.4 Mapping example . . . 7
- 2.3 State-of-the-art on the mapping of quantum circuits . . . 10
- 2.4 Mapping metrics . . . 11
  - 2.4.1 Number of SWAPs . . . 11
  - 2.4.2 Latency . . . 11
  - 2.4.3 Algorithm reliability . . . 11

3 Mapping on superconducting quantum processors **13**
- 3.1 Constraints of the Surface-17 chip . . . 13
  - 3.1.1 Superconducting quantum processor architecture . . . 13
  - 3.1.2 Gate set, gate time and gate fidelity . . . 14
  - 3.1.3 Qubits interaction constraint . . . 16
  - 3.1.4 Frequency constraint . . . 16
  - 3.1.5 Measurement constraint . . . 17
- 3.2 Mapping model . . . 18
  - 3.2.1 Initial placement . . . 19
  - 3.2.2 Router . . . 19
  - 3.2.3 Scheduler . . . 19
- 3.3 New Mapping Metrics: algorithm's reliability . . . 19
  - 3.3.1 Fidelity and Probability of success . . . 19
  - 3.3.2 Quantum Volume . . . 20

4 Benchmarks and analysis framework **23**
- 4.1 Benchmarks . . . 23
  - 4.1.1 Benchmark selection . . . 24
- 4.2 Analysis framework . . . 24
  - 4.2.1 Compiler (OpenQL) . . . 24
  - 4.2.2 quantumsim . . . 25
  - 4.2.3 Analysis Framework . . . 25

5 Results **29**
- 5.1 Impact of the mapping on the algorithm reliability . . . 29
- 5.2 Analysis of the mapping metrics . . . 31

6 Conclusions and Future Work **39**
- 6.1 Conclusions . . . 39
- 6.2 Future work . . . 39

Bibliography **49**

1 Introduction

1.1. Motivation

The discoveries in quantum physics during the last century opened multiple questions due to the observation of extraordinary behaviours that are far from the classical physics we are used to. The answers of these questions as well as another field's investigations rely on the simulation of Nature. In order to understand our world we need to simulate instances of it, but the simulation of Nature itself requires a huge computational power that is intractable for classical computers. At the same time, we are witnesses of the Moore's law halt; we are reaching the limit in transistor sizes. While trying to make them smaller, quantum phenomena appear making impossible a transistor behaviour. During the last century Feynman had the idea of getting advantage of the unexplained but controllable quantum phenomena. He noticed that this quantum phenomena that would stop the development of smaller transistors and, therefore, powerful chips, actually could be used to get much more computational power. He was the first to introduce the idea of building a quantum computer to simulate quantum mechanics in 1982.

Since then, research on quantum computing has been mostly focusing on the development of quantum technologies and quantum algorithms. In order to bridge the gap between them, the Quantum Computer Architecture Lab at Delft University of Technology proposed a full-system stack, an architecture for a quantum computer [1]. As shown in Figure 1.1 it consists of quantum algorithms that can be expressed expressed by a high-level programming language and then compiled to a series of instructions. In the microarchitecture, those instructions are translated into signals that are sent to operate on the qubits.

As a part of this full-system stack, and when targeting a real quantum processor, quantum algorithms need to be adapted to the constraints of the quantum chip; they need to be mapped.

1.2. Problem Statement

Quantum algorithms are represented as quantum circuits when the circuit model of computation is adopted. Quantum circuits consist of quantum bits and gates operating on them. This representation (at high-levels of the system stack) is hardware agnostic; that is, it does not take into account the possible constraints of the quantum chip. Therefore, mapping models are required to have a version of the quantum algorithm adapted to the quantum device and that can be executed on it. Note that, one of main constraint in current quantum chips is the reduced connectivity between the qubits that limits their interaction to only nearest- neighbours.

The mapping process results in an increase of the number of gates of the circuit and/or the circuit depth (or circuit latency), affecting the reliability of the algorithm. As qubits decohere and gates are error prone, the higher the number of operations or the higher the circuit depth, the higher the probability of having an error during computation. Therefore, mapping models require an optimization process in search of the best circuit version that is executable on a given device and still produces 'good' results.

Most of the works done about the mapping task optimize in terms of two parameters, either the number of

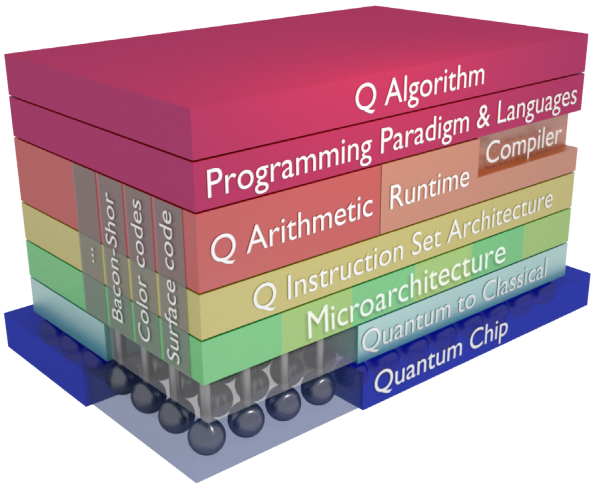


Figure 1.1: Full stack implementation [2]

added operations in the adapted version of the circuit or the latency added to the circuit. Moreover, these works asses the quality of their mapping algorithms based on one of those two metrics. It is clear that both parameters should be as small as possible. However, these metrics are not complete and do not give nay information on how the mapping affects the algorithm's reliability.

Given this scenario, the aim of this thesis is to propose different metrics that show how the algorithm's reliability is affected by the mapping process. To this purpose, we will focus on the fidelity and the probability of success and how they relate with the quantum volume and other metrics such are number of gates, number of two-qubit gates and circuit depth.

1.3. Structure of the thesis

This thesis follows the structure described below:

In chapter 2, a background on the quantum computing field is given. We also explain the insights of the mapping task, as well as current state-of-the-art in this field. Finally, a description of the most used mapping metrics is described.

In chapter 3, we describe the constrains of the quantum chip that we target to map quantum algorithms on. Constrains that we need to take into account in our mapping model, which we describe afterwards. This chapter concludes with an explanation of the new mapping metrics we will investigate.

In chapter 4, we explain the process we followed to evaluate the mapping metrics. First, we select a set of quantum algorithms as benchmarks. This benchmarks will be mapped following the constrains of the quantum chip and then simulated. This process will be led by the analysis framework we developed and that we describe in detail in this chapter.

In chapter 5, the results are presented and, finally, in chapter 6, we conclude the thesis and we set the future work for this topic.

2
Quantum computing and mapping of quantum circuits

In this chapter we offer the background in Quantum Computing required throughout the thesis. First, in section Quantum computing we describe the general rules and the basic elements of a quantum computer. After that, in the Mapping of quantum circuits section we explain what is the process of mapping quantum algorithms, why is required and the basic steps on it. In the State-of-the-art on the mapping of quantum circuits we talk about the current works and improvements done during the last years about the mapping task. We finish the chapter with a description of the metrics used to measure the quality of the mapping process and that serve as parameters to optimize in the mapping algorithms, in the Mapping metrics section.

2.1. Quantum computing

Quantum computers will be able to solve problems that are intractable for classical computers. Due to the quantum phenomena, quantum computers are able to explore a whole solution space at once. The most famous example is Shor's quantum algorithm. It is an integer factorization algorithm able to calculate prime numbers in polynomial time, which is exponentially faster than its classical counterpart, General Number Field Sieve (GNFS). In general, quantum computers are a promising technology able to solve problems considered hard in classical computation. In the next sections we will explain the basics of quantum computation.

2.1.1. Essential elements of quantum computation

Quantum bits (or qubits) and quantum gates are the basic ingredients in quantum computation. As in classical computation with boolean gates and bits, quantum gates operate on the qubits to change their state and return the desired result or calculation.

1. Qubits

   A qubit is the basic unit of information in quantum computing. A classical bit can be either 0 or 1. A qubit's state could be either 0 or 1 or both at the same time, what is called **superposition**. Both, ground or excited states, are represented in the Dirac's bra-ket notation [3], $|0\rangle$ and $|1\rangle$ respectively. A qubit is described as in a probabilistic state $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$, where $\alpha_0, \alpha_1 \in \mathbb{C}$ are the so-called probability amplitudes. $|\alpha_0|^2$ is the probability of measuring $|0\rangle$ and $|\alpha_1|^2$ is the probability of measuring $|1\rangle$. In other words, when a qubit is measured, one can get the measurement result '0' with probability $|\alpha_0|^2$ and '1' with probability $|\alpha_1|^2$. In addition, the qubit's state collapses to either $|0\rangle$ or $|1\rangle$, respectively. As $|\alpha_0|^2$ and $|\alpha_1|^2$ are probabilities, the sum of them should be $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Therefore a qubit state can be represented as a vector (eq. 2.1).

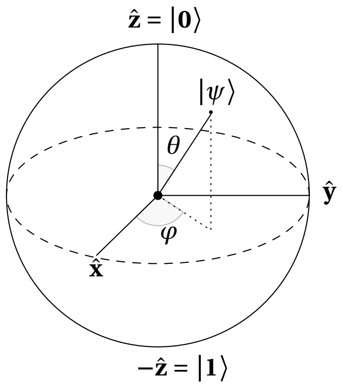

$\hat{\mathbf{z}} = |\mathbf{0}\rangle$
$|\psi\rangle$
θ
$\hat{\mathbf{y}}$
φ
$\hat{\mathbf{x}}$
$-\hat{\mathbf{z}} = |\mathbf{1}\rangle$

Figure 2.1: The Bloch sphere

$$|\psi\rangle = \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} \tag{2.1}$$

As soon as the vectors are 2-dimensional, complex and unitary they can also be described in phase notation [3]. Like the complex numbers, although requiring two angles in this case. This notation lead us to an understandable way to visualize the quantum states, the **Bloch sphere** (Fig. 2.1). A sphere of radius one, which z-axis extremes represent the ground and excited states.

The power of quantum computers comes from the combination of several qubits. The quantum state of n qubits can be represented in bra-ket notation, with a 2^n size. For instance, the state of two and n qubits are represented in the eq. 2.2 and 2.3, respectively. Another important quantum property is **entanglement**, which qubits are correlated with each other.

$$|\psi\rangle = \alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle \tag{2.2}$$

$$|\psi\rangle = \alpha_0|0...0\rangle + \alpha_1|0...1\rangle + ... + \alpha_{n-1}|1...1\rangle \tag{2.3}$$

2. Quantum Operations

Quantum operations leverage the power of quantum computers enabling calculations on the qubits. Quantum operations change the state of the qubit. We consider three kind of operations in quantum computing: gates, qubits' measurements and qubit initialization processes. Quantum gates are represented as square matrices of $2^n \times 2^n$, where n is the number of qubits involved in the operation. This matrices should be unitary respecting the qubit state vector unitary property ($|\alpha_0|^2 + |\alpha_1|^2 = 1$). Single-qubit operations can be represented as state rotations in the Bloch sphere (Fig. 2.1). For instance, an X gate is a rotation of 180° in x-axis of the Bloch sphere and changes the qubit state from $|0\rangle$ to $|1\rangle$, or viceversa. An example of the matrix representation of a quantum operation and the way to operate with qubits is shown in eq. 2.4.

$$U|\psi\rangle = \begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \begin{bmatrix} \alpha_0 u_{00} + \alpha_1 u_{01} \\ \alpha_0 u_{10} + \alpha_1 u_{11} \end{bmatrix} \tag{2.4}$$

(a) Single-qubit gates

Single qubit gates operate just on one qubit. As explained before, single-qubit gates can be represented as $2^1 \times 2^1$ square matrices that should be unitary. The symbol and the matrix representation of the most common single-qubit gates can be found in Table 2.1.

The *Identity* gate is the idling operation. It is equivalent to no applying any operation for a cycle. The *Pauli-x, -y and -z* gates are 180° rotation over the x-, y- and z-axis, respectively. The *Hadamard*

Table 2.1: Most common single qubit gates

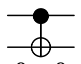
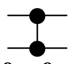
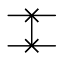
Identity	Pauli-X	Pauli-Y	Pauli-Z	Hadamard	S gate	T gate
$-\boxed{I}-$	$-\boxed{X}-$	$-\boxed{Y}-$	$-\boxed{Z}-$	$-\boxed{H}-$	$-\boxed{S}-$	$-\boxed{T}-$
$\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$	$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$	$\begin{bmatrix}0 & -i\\i & 0\end{bmatrix}$	$\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$	$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$	$\begin{bmatrix}1 & 0\\0 & i\end{bmatrix}$	$\begin{bmatrix}1 & 0\\0 & e^{i\pi/4}\end{bmatrix}$

gate is also a 180° rotation, but over the diagonal axis between the x- and z-axes, $\frac{(\hat{x}+\hat{z})}{\sqrt{2}}$. The *S* and *T* gates are also rotations over the z-axis but of 90° and 45° respectively.

(b) Two-qubit gates

Two-qubit gates are quantum operations that involve two qubits. In general, the two-qubit gates execute a single-qubit operation over one of the qubits, depending on the state of the other. The qubits that goes through the operation is called **target**, while the other is called the **control** qubit. The most common two-qubit gates are represented in Table 2.2. The *CNOT* gate is a Controlled-NOT operation or, what is the same, a Pauli-x gate that, depending on the state of the control qubit will be executed or not. If the control qubit is $|1\rangle$, a Pauli-x will be executed onthe target qubit. As the CNOT, the *CZ* gate is a Controlled-Z operation that, in this case, performs a Pauli-z gate in the target qubit when the control qubit is $|1\rangle$. Finally, the *SWAP* gate exchanges the state of two qubits. This gate is mostly used for routing purposes as it will be seen in the next sections.

Table 2.2: Most common two-qubit gates

CNOT	CZ	SWAP
		
$\begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & -1\end{bmatrix}$	$\begin{bmatrix}1 & 0 & 0 & 0\\0 & 0 & 1 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\end{bmatrix}$

(c) Universality

A **universal set of gates** is a set of operations able to generate any other gate by combining them [3]. In classical computation, for example, the `OR` and the `AND` gates are able to generate any other logic gate. Also comparable with the boolean gates, quantum operations can be decomposed in other set of quantum operations. In quantum computation there are several universal set of gates. The most used one is the **Clifford+T** set, formed by the Clifford gate set – phase shifts around the three axes, H and CNOT – and the T gate.

2.1.2. Quantum Circuits

Quantum algorithms can be described by quantum circuits when the circuit model of computation is adopted. As mentioned before, they consist of quantum gates and qubits connected in circuit fashion. As most of the algorithm description models – no matter if classical or quantum –, quantum circuits are hardware agnostic, which is that they are not specified to any quantum device. In Fig. 2.2 we present an example of a quantum circuit. This circuit represents the quantum equivalent of a Gray encoder of six bit length. It is composed by 6 qubits and CNOT gates.

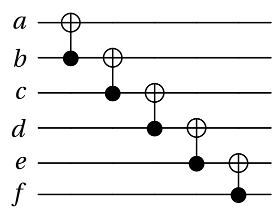


Figure 2.2: Gray encoder quantum circuit.

2.1.3. Qubits are faulty

Quantum operations are faulty and qubits are not able to hold the desired state for long times, gradually rotating to another state – the qubit decoheres. For instance, in the case of superconducting technologies [4], the chips bear with decoherence times of ~ $30\mu s$ for qubit relaxation and ~ $60\mu s$ for qubit dephase. The error rates of single-qubit gates are less than 0.1% taking > $20ns$ to be executed, while two-qubit gates error rate is 0.6% with times of $40ns$ and measurement error rates around 1% with execution times of ~ $300ns$ [4, 5]. This creates an undesirable environment to compute the most useful algorithms. Therefore, in order to fight the errors generated by this behaviour, fault-tolerant (FT) and quantum error correction (QEC) mechanisms have been developed during the last years [3].

2.2. Mapping of quantum circuits

As it was described in the Motivation section, the mapping step is a critical part of the process to run quantum algorithms. Quantum algorithms can be described by quantum circuits, that are hardware agnostic and assume any interaction between qubits possible. They assume the qubits are all-to-all connected. However, quantum chips have limitations (see Constraints of the Surface-17 chips). One of the main constraints is the Nearest Neighbor (NN) constraint. Instead of connected in an all-to-all fashion, the qubits are arranged in a two-dimensional grid in which connectivity between qubits is limited; each qubit connects at most with four neighbours. Then, the circuits need a transformation in order to be executed in real quantum device. This transformation is the so-called mapping process.

The mapping process adapts a quantum circuit representation to the requirements of a real quantum processor. It brings ideal algorithms down to earth, to the limitations of the quantum chips. Therefore, the mapping accounts for the realization of quantum algorithms in quantum chips. In order to do this, the qubit states need to switch places between them – introducing SWAP gates – whenever an interaction between two qubits that are not NN is required. But, as explained in the previous section (Qubits are faulty), quantum gates are faulty and the longer a circuit is, the more errors appear. Then, the extra addition of gates or the circuit depth increment due to the mapping task, causes quantum algorithms to accumulate errors and results in more noisy or even useless results. Therefore, the optimal mapping would be the one that alters the least the original circuit. In order to address the mapping problem, we split it in three steps: **scheduling**, **initial placement** and **routing**. These steps can be executed several times, or even continuously, and in any order, depending on the mapping approach.

2.2.1. Initial placement

Throughout this thesis we use the terms ***virtual*** **qubits** and ***physical*** **qubits** referring to the qubits from the circuit that will be mapped to the real qubits of the quantum chip, respectively. The initial placement task is responsible to relate the virtual qubits with the physical ones, trying to find the best arrangement in order to minimize non-NN interactions. As we will see in the example below, an optimal initial placement may help to reduce the number of SWAP gates introduced by the routing step.

2.2.2. Routing

When two qubits that need to interact – i.e. perform a two-qubit gate – are not NN, they have to be moved or 'routed' to adjacent positions. The routing process accounts for finding the best path -e.g. shortest path- between two qubits far away in the chip that need to interact. Qubits (quantum states) can be moved from one position to another by means of SWAP gates. Therefore, it is a decisive step in order to reduce the number of added gates to the circuit and the circuit depth.

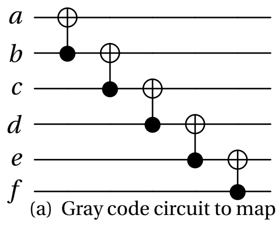
2.2.3. Scheduling

A scheduler organizes the circuit operations through the circuit time, finding whether several operations can be executed in parallel – at the same time – or not. It is possible to schedule in different configurations, for instance As Soon As Possible (**ASAP**) or As Late As Possible (**ALAP**), depending on the mapping requirements.

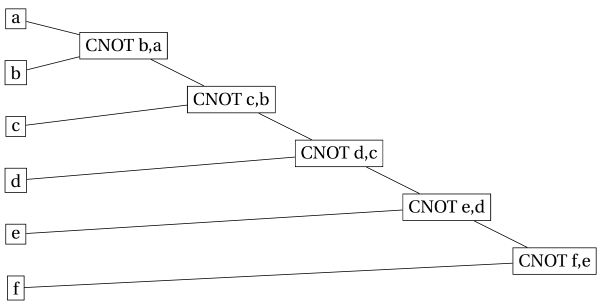
In order to schedule the operations, a dependence graph is built. A graph that relates qubits and gates sequentially. Starting from the qubits, it chains all the gates with them. It is really convenient while mapping, mostly scheduling and finding the parallel operations. The gates that are in the same graph column can run in parallel.

2.2.4. Mapping example

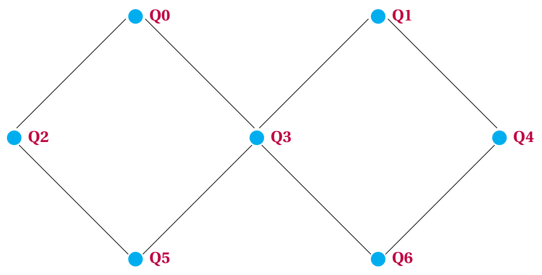
In order to illustrate the mapping steps let us consider the following circuit (see Fig. 2.3(a)) and the device's chip layout in Fig. 2.3(c), in which circles represent the physical qubits and the edges the connections between them and then possible interactions. We want to map the Gray code circuit to such as chip layout. The mapping flow that we will follow starts with a scheduler, then we set the initial placement, route and, finally, we re-schedule in order to make all the gate additions as parallel as possible. As illustrated in the dependence graph of Fig. 2.3(b), there are no parallel gates. No better schedule is possible. Considering the gate times for this device the same ones as the ones described in Tab. 3.2, the latency of this circuit is 400ns already.



(a) Gray code circuit to map



(b) Dependence graph of the circuit



(c) Chip layout where to map the example circuit

Figure 2.3: Mapping example draft

1. Initial placement

If we place the qubits based on the required interactions – that is: a next to b, b next to c and so on – we can find in this case a placement in which all qubits can perform the required two-qubit gates as shown in Figure 2.4. Then, with an optimal initial placement, we can observe that the circuit could be the same without adding any gate. No routing would be required.

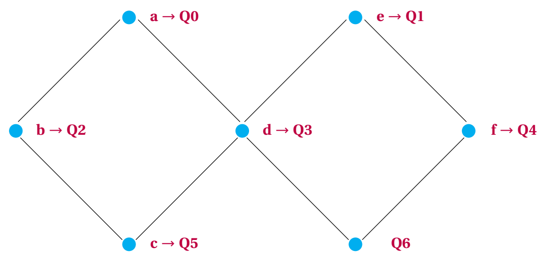


Figure 2.4: Chip layout with the qubits in the optimal initial placement

However, if we do a naive initial placement in which, for instance, the virtual qubits ($a, b, c \dots$) are mapped to the physical ones $Q_0, Q_1, Q_2 \dots$ in alphabetical order (see Fig. **??**). We will need to start routing them.

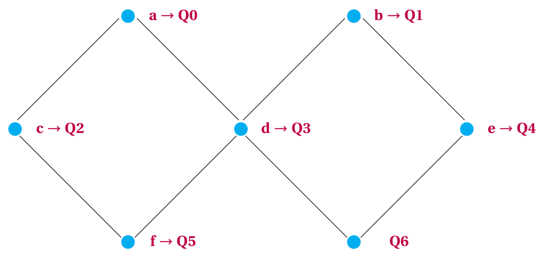
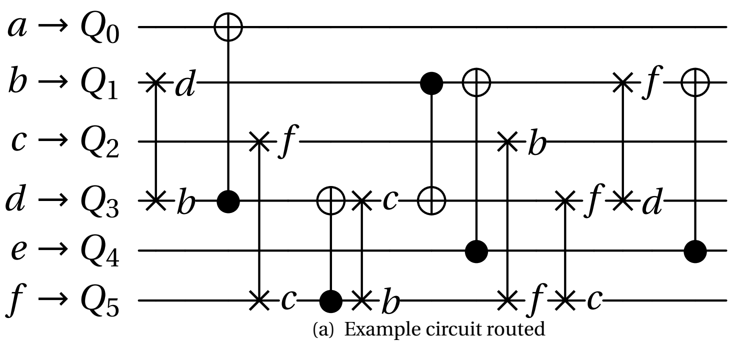
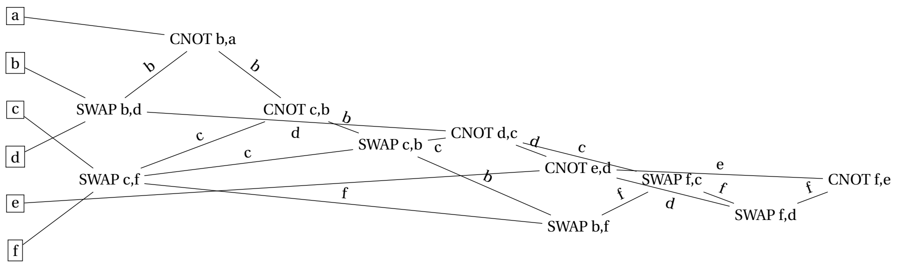


Figure 2.5: Qubit disposition in the chip layout with a naive initial placement

2. Routing

In order to allow the non-NN qubits to interact, we need to move them to adjacent positions by inserting SWAPs. The mapping brings smartly the qubits close to the next operation qubit as well. For example, it routes b next to a but also close to c, avoiding a path increment for the next operation. The result can be seen in Fig. 2.6(a). After every SWAP, the interchanged virtual qubits appear to follow easily the mapping. The latency after the routing increases in 1440 ns, 1440 + 400 = 1840 ns.


(a) Example circuit routed



(b) Dependence graph after routing

Figure 2.6: Naive initial placement after routing

3. Scheduling

   As one could notice in the dependence graph (Fig. 2.6(b)), there are several operations that can be parallelized. Therefore, we can apply a scheduling. In the Fig. 2.7 the result of an ASAP scheduling is shown, reducing the latency to 1520 ns. Note that the dashed boxes enclose all the parallel operations in a cycle.

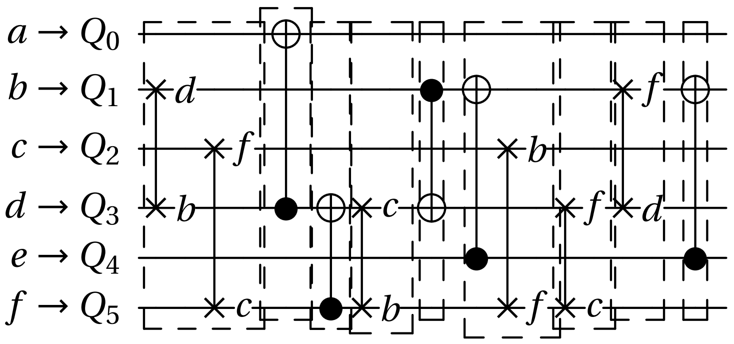


Figure 2.7: Naive initial placement routed and re-scheduled

We summarize the results of using a naive and an optimal initial placement in terms of number of gates and latency in Tab. 2.3.

Table 2.3: Difference between the naive initial placement and the optimal one in terms of number of operations and latency

	Optimal approach	Naive apprach
# operations	5	11
latency	400 ns	1520 ns

2.3. State-of-the-art on the mapping of quantum circuits

Quantum algorithms are meant to leverage the promising power of quantum computers [6]. Commonly described as quantum circuits, quantum algorithms are hardware agnostic. However, as we have seen they need to be adapted – mapped – to the quantum device limitations. The main constraints in all of them is the limited connectivity between qubits. There is a vast amount of literature on the algorithms – high abstraction level –, neglecting hardware constraints. From ion traps [7] to superconducting qubits [5, 8], through quantum dots [9, 10], each layout has its own requirements and constraints. Although all of them are arranged in a 2D – planar – structure, ion traps technologies are capable to connect the qubits in all-to-all networks while superconducting technologies connections form a grid shaped network where each qubit is connected to a maximum of four neighbors. This grid structure establishes one of the main connectivity limitation in today's quantum chips, the nearest-neighbour constraint. Also industry's superconducting chip layouts from IBM [11], Google [12] and Rigetti [13] follow this structural limitations. As for the high abstraction level of the quantum algorithm, a link between the algorithms and the devices is required [1]. As in classical computation, the algorithms should go through a transformation process in order to adapt them to the hosting device. Certainly, the mapping procedure is an important element of this process.

There is a considerable amount of literature on the mapping task. Initial works on this field [14–16] focused primarily on the definition of what they characterized as a *scheduler* able to parallelize operations and add the required gates to route qubits. They consider general connectivity constraints as the NN one, common for most of the devices – although the works were examining ion-traps as hardware implementations. The proposed techniques by these works examine a dependency graph looking for the best way to organize qubits and operations. The majority of the methods use latency as the metric to minimize, however some of them [17] would minimize in number of SWAP operations. As it will be explained in the next section, to minimize in latency stands for time optimization while minimize in number of SWAP operations means to find the path that introduces the minimal number of SWAP operations. Following a similar reasoning as the first approaches, more complex solutions [18] have been published using Constraint Programming together with temporal planning, optimizing in latency as well. Also, several publications [19, 20] outline only the routing sub-task using the number of SWAPS as the metric to minimize.

A recent review of the literature on the mapping topic focused on mapping quantum algorithms on a specific quantum device. In all the cases, taking into account the specific chip connectivity constrain. IBM's chip have been gaining much attention due to the open online tools that make the chips accessible to everybody. Various approaches to map algorithms for the IBM family of chips have been proposed [21–24]. Zulehner et. al [21] developed a routing algorithm that optimizes in the number of SWAPs building a graph – similarly to previous works – and searching the best route in the chip layout with the A* algorithm. Siraichi et. al [22] work defines a weighted dependence graph where the mapping algorithm is able to find a solution for all the mapping steps, initial placement, scheduling and routing. Also, apart from the works related to the IBM devices, Rigetti's devices have been approached [25].

Many attempts have been made [26–30] with the purpose of develop a FT mapping able to work at the logical – qubit – level. However, due to the high complexity of the QEC techniques, quantum chips with large amounts of qubits are still theory. Current and near-future quantum processors are what Preskill [31] calls Noisy Intermediate-Scale Quantum (NISQ) devices; chips with an amount of 50-100 qubits and without QEC or much simpler encodings. Several studies, for instance [32–34], have been conducted on the mapping algorithms required for NISQ devices. Also, the works on device specific solutions [21–25] described before, could be considered part of the NISQ works collection.

Besides latency and the number of operations that serve to analyze the quality of a mapping algorithm, few studies have been published on quantum metrics. In this thesis, we will propose another metrics to evaluate the mapping process. Fidelity has been addressed in order to characterize the error of a quantum circuit based on the qubits state [3, 35]. And, recently, Quantum Volume has been defined to assert the capability of a quantum computer [36]. In the next section and in the New metrics section we offer an overview of the important metrics in this work.

2.4. Mapping metrics

As outlined in the state-of-the-art, the literature considers mainly two metrics to optimize in their mapping algorithms and as an output to evaluate how good the mapping is. These are the added **latency** and the **number** of introduced **operations** after the mapping procedure. The routing step from the mapping process introduces SWAP gates in the quantum circuit in order to move the qubits around. And latency is the time required to run a quantum circuit in a given quantum device. It depends on the chip cycle time and the **depth** of the circuit, $Latency = depth \times t_{cycle}$. Depth is the number of cycles the algorithm encloses. Actually, most of the studies [14–16] assert the latency in terms of circuit depth, avoiding the cycle time specification of any device. As shwon in Fig. 2.7, one can see how after some mapping process some SWAP operations have been added as well as the depth of the circuit or latency grows. From five cycles depth in the original circuit to nine cycles.

2.4.1. Number of SWAPs

As explained before in the Mapping of quantum circuits section, routing means to introduce SWAP operations – or any other operation that allows qubits to be moved. Whenever a two-qubit gate requests two qubits that are far apart, a path in the gridlike chip layout should be given. Each step on this grid would mean a SWAP operation between the vertex qubits of that step. We already know that quantum gates are error prone and that one of the main problems that the mapping task is dealing with is to use the least amount of gates as possible. Although the general mapping problem is to avoid the introduction of error to the original algorithm, that is caused by more sources apart from the number of gates. Nonetheless, the quantum gates is one of the main error sources altogether with the decoherence time. That is the reason why the number of SWAPs is commonly used to assert the quality of a mapper, but it is not enough.

The number of SWAPs does not give any information about how long the circuit is, or how much it will be affected by decoherence. The higher number of SWAPs does not mean the longer the circuit necessarily. The SWAPs can be in parallel, for instance. It could be the case of a short circuit with a long number operation in parallel.

2.4.2. Latency

Latency is the time required to run a quantum circuit in a given quantum device. The introduction of gates due to the mapping makes the target circuit longer. Or what is the same, the introduction of SWAPs adds several cycles to the depth of the circuit. Therefore, the latency of the circuit increases. For instance, in the Mapping example, we can see how the latency grows after the circuit is routed to 1840 ns and then it is reduced to 1520 ns after re-scheduling.

One of the main source of errors in quantum computing is the decoherence, that is the quantum phenomena that makes the qubits loose their state along time. The longer a quantum device is running an algorithm, the more errors would appear. Or what is the same, the longer the circuit, the more erroneous will be the result. The task of the scheduler – one of the three mapping steps – is to look for gates that can be run in parallel. Considering that time is one of the main error sources and that the scheduler task is to make operations simultaneously, one can understand why latency is used in order to assert the quality of a mapping algorithm. Nonetheless, as the number of SWAPs this metric is not giving enough information for that.

The latency of a circuit gives an intuition of how long it is but, it is not sufficient to assess the quality of the mapping. For instance, unlike the number of SWAPs metric, it gives no information about the number of operations a circuit has.

2.4.3. Algorithm reliability

The ideal mapping would be the one affecting the least as possible the original circuit or, what is the same, introducing the least amount of errors. Clearly, both, latency and number of SWAPs, are direct effects of the mapping procedure that have an impact in the error increase as well. To optimize in any of them stands on correct – but not sufficient – reasoning. Latency optimization assumes that the most important error source comes from the qubit lifetime. And, indeed, time is a main issue in quantum computing. Decoherence time is not only a maximum qubit lifetime, it is harder and harder to hold a quantum state for use times closer to the decoherence time. On the other hand, to optimize in number of SWAPs stands on the intuition that the gate error is the prominent error in a quantum device. Although intuitively one can think that the more operations are in a circuit, the longer it will be; as seen in the Problem Statement section, the higher the number of operations does not necessarily mean the longer circuit depth. Several operations can be run in parallel in the same cycle – at the same time. Actually, to minimize in latency is to maximize in parallelism. Therefore, even though related, to optimize in one or the other holds different baselines.

In addition, these metrics are not enough to assess how good the mapping process is. These metrics do not tell you how the algorithm is affected by them. In other words, will my algorithm still produce 'good' results after the mapping? They are related with the error rate increase but the relationship is not totally clear. For instance, it could be the case that concatenation of errors introduced by contiguous gates would end up in a correction of the first error. Or, that the busier the qubit is, the less affected by decoherence would be.

In this thesis, we propose three new metrics to analyze how the mapping process affects the algorithm reliability. As we will explain in Section New Mapping Metrics: algorithm reliability these metrics are fidelity, probability of success and quantum volume. To perform this analysis we will map several small quantum algorithms on the superconducting quantum chip develop at Qutech by DiCarlo's group [5]. In the next chapter, we will explain the constraints of this chip and the mapping model that we used.

3
Mapping on superconducting quantum processors

In this chapter we will describe the superconducting quantum processor used in this thesis and its constraints. Although the work done in this thesis could be apply to any quantum technology, we limit our study to the surface code 17 chip (Surface-17) developed by Leo Di Carlo's group at QuTech [5]. In addition, we will explain the mapping model and the new metrics proposed in this work.

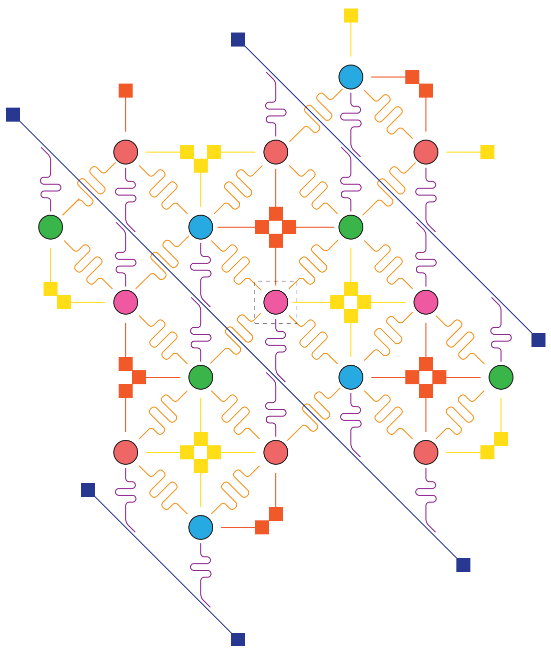
3.1. Constraints of the Surface-17 chip

In this section the constraints of the superconducting Surface-17 quantum processor developed at QuTech by Di Carlo's group are summarized. To operate on this kind of qubits, analogue signals with a specific frequency, amplitude and envelop are sent.

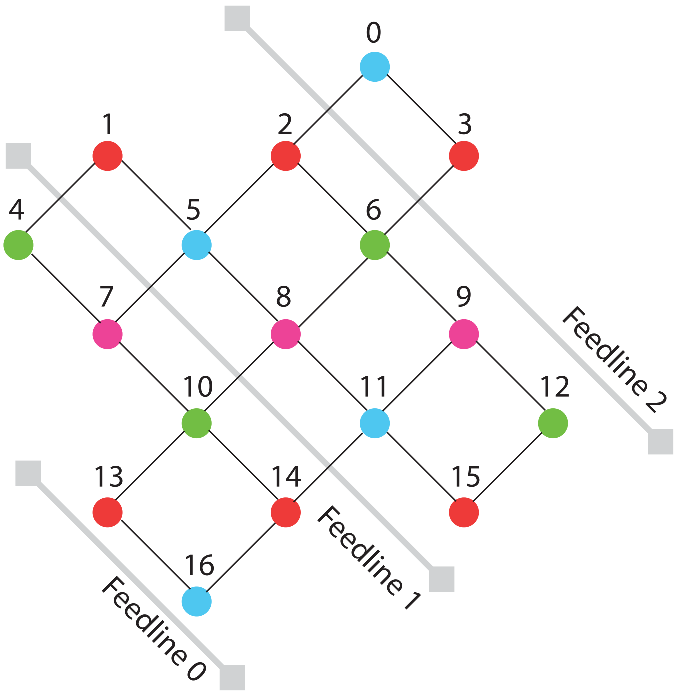
3.1.1. Superconducting quantum processor architecture

Figure 3.1(a) shows a schematic of the layout for the Surface-17 chip where the circles represent the qubits [5] and the different colors the frequency used for single-qubit microwave control. In order to perform a single-qubit gate microwave pulses are used, whereas flux-pulses (qubit specific detuning sequence) are used for two-qubit gates. To this purpose, every qubit has a dedicated flux-control line (yellow) and microwave-drive line (red) to control the qubit frequency and send operating waves, respectively. Several qubits are connected through a readout resonator (in purple) to the feedlines (diagonal blue lines) that are used for measuring them. Note that readout resonators are simultaneously interrogated using frequency-division multiplexing. Finally, only qubits connected by bus resonators (in orange) can interact, this is the nearest-neighbour constraint. For more details, please read [5].

Figures 3.1(b) show a schematic of the topology for the Surface-17 quantum chip. Circles represent the **qubits** and the **edges** represent the 'connections' or possible interactions between them. Numbers are the physical addresses of the qubit. As we just mentioned the different colors represent the frequency for single-qubit microwave control: red for f_1, blue and green for f_2 and pink for f_3.



(a) Schematic of the realization of SC-17 chip [5]



(b) Simpler layout of the SC-17. All the examples will refer to this picture

Figure 3.1: SC-17 chip layouts describing its architecture

3.1.2. Gate set, gate time and gate fidelity

In order to perform universal computation, a whole set of universal gates should be executable in a given device. However, not all kind of gates are supported in real quantum processors. In our case, any kind of single qubit rotation can be performed on the superconducting quantum processor, but gates need to be calibrated before performing any experiment and obviously one cannot calibrate infinite amount of gates. Furthermore, Z rotations do not perform very well on this kind of superconducting qubits. Based on these observations, we will limit single qubit gates to X and Y rotations, in which the most common are ± 90 (degrees) and ± 180. Conditional-phase (CZ) gates and measurement in the Z basis are also possible. Therefore, the gates supported (elementary gates) by the superconducting quantum processor are:

- X gate with arbitrary angle rotation. Most common ±90 and ±180 degrees rotations
- Y gate with arbitrary angle rotation. Most common ±90 and ±180 degrees rotations
- CZ gate
- Measurement in the Z basis (M_z)

Table 3.1 shows the gate time and the errors rates for single-qubit gates, CZ gate and measurement [5]. Note that the default basis of measurement will be Z basis unless one specifies.

Table 3.1: The gate time and fidelity of primitive set from experiments. The measurement time includes both measuring and depletion time.

Gate type	**Gate time**	**Fidelity**
Single-qubit	20 ns	~ 99.97%
CZ	40 ns	~ 99.93%
M_z	600 ns	~ 99.5%

For the mapping process, we assume that arbitrary rotations and multi-qubit gates are first decomposed to the Clifford+T group, {H,T,S,CNOT}. Although this is a universal gate set that is not directly supported by the quantum chip. These gates need to be further decomposed into the elementary gates as shown in Figure 3.2.

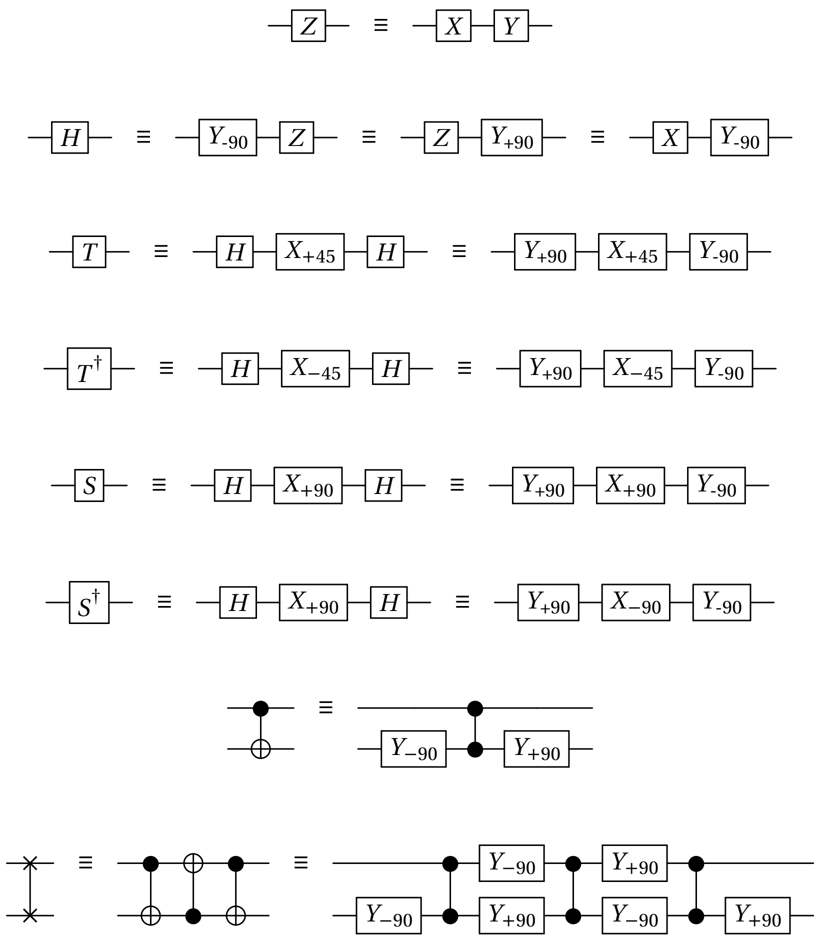

Z ≡ X Y
H ≡ Y-90 Z ≡ Z Y+90 ≡ X Y-90
T ≡ H X+45 H ≡ Y+90 X+45 Y-90
T† ≡ H X−45 H ≡ Y+90 X−45 Y-90
S ≡ H X+90 H ≡ Y+90 X+90 Y-90
S† ≡ H X+90 H ≡ Y+90 X−90 Y-90
≡ Y−90 Y+90
≡ ≡ Y−90 Y−90 Y+90 Y+90 Y−90 Y+90

Figure 3.2: Decomposition in the X-Y-Z-CZ set of the Clifford + T gates that are not directly supported in the quantum chip

Their gate time – shown in Table 3.2 – depends on the gate time of the gates in which is decomposed. We show this gate times, the gate time of the chip allowed gates, in Table 3.3.

Table 3.2: The gate time for the universal set {H,S,T,CNOT}.

Gate type	Gate time
X	20 ns
Y	20 ns
Z	40 ns
H	40 ns
S/Sdag	60 ns
T/Tdag	60 ns
CNOT	80 ns
SWAP	200 ns
M_Z	600 ns

Table 3.3: The gate time for the universal set allowed in the devices $\{R_X, R_Y, CZ, M_Z\}$

Gate type	Gate time
$R_X(\pm 45, \pm 90)$	20 ns
$R_Y(\pm 45, \pm 90)$	20 ns
CZ	40 ns
M_Z	600 ns

.

In the next sections, the different constraints of the superconducting quantum chip will be explained in detail. Note that we will distinguish two kind of constraints: the ones coming from the quantum chip- e.g. nearest-neighbour interactions- that we call **hardware constraints** and the ones from the electronics control setup - e.g. qubits operated by the same AWG- called **electronics constraints**.

3.1.3. Qubits interaction constraint

In Surface-17 each qubit is connected to a maximum of four qubits, the nearest-neighbours, limiting the possible interactions -e.g. two-qubit gate- between them (hardware constraint). This constraint will make that qubits that need to interact and are not place in neighboring positions will need to be moved to adjacent positions. Quantum states in superconducting technology can be 'moved' by using SWAP operations.

This constraint implies that in Surface-17 (see Figure 3.1(b))

It is possible to do:

```
CZ q[1],q[5]
CZ q[8],q[6]
```

but impossible to do (directly, without any mapping):

```
CZ q[1],q[2]
CZ q[0],q[16]
```

The code shown here and in the next sections is written in an eQASM fashion. The letters represent the kind of gate operation and the number will refer to the qubit number in the Surface-17 layout (Figure 3.1(b)). Also, the character '|' represents the operations that can be executed in parallel.

3.1.4. Frequency constraint

Single-qubit gates

In order to perform single-qubit gates, electromagnetic microwaves are sent. As shown in Figures 3.1(b), three or four different frequencies for single-qubit microwave control can be used in Surface-17. In principle, any qubit can be operated individually and then any combination of single-qubit gates can be performed in parallel. However, qubits using the same frequency are controlled by the same Quantum Waveform Generator (QWG) and then they are limiting the possible parallelism of operations.

Table 3.4: Frequency groups for Surface-17 when using 3 different frequencies.

QWG	Qubits	Frequency Group
0	1, 2, 3, 13, 14, 15	f_1
1	7, 8, 9	f_3
2	0, 4, 5, 6, 10, 11, 12 16	f_2

It is not possible to perform two different single-qubit gates at the same time on qubits using the same frequency and operated by the same QWG. For instance in SC-17,

It is possible to do:

```
{ X q[1] | X q[7] }
8{ X q[6] | Y q[7] }
{ X q[1] | X q[2] | X q[5] | X q[0] }
{ Y q[10] | Y q[4] }
```

But not

```
{ X q[1] | Y q[2] }
{ X q[5] | Y q[0] }
{ X q[10] | Y q[4] }
{ X q[7] | Y q[9] }
```

Obviously, it is not possible to perform more than one single-qubit gate on the same qubit at the same time (gate dependency).

Two-qubit gates

In order to perform a two-quibt gate in superconducting qubits, interacting qubits need to be brought to a specific close frequencies [5]. More specifically, CZ_f gates are performed between qubits at f_1^{int} and f_2 or at f_2^{int} and f_3 as shown in Figure 3.3. In addition, the qubits that share a connection with the qubits performing a 2-qubit gate need to be detunned away. This limits the amount of two-qubit gates that can be performed in parallel (hardware constraint).

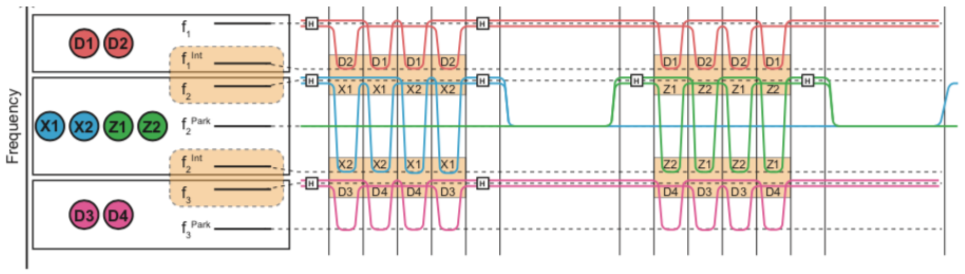


Figure 3.3: Frequency arrangement and detuning frequencies for the qubits in the unit cell.

As we assume the frequency scheme of Figure 3.3 we can formulate the following: 1) If a two-qubit gate is being performed between D1 (D2) and X1 (X1 or Z1), D2 (D1) cannot interact with X1 or Z1 (X1) at the same time; 2) In the same way, if a two-qubit gate is being performed between D3 (D4) and X1 or Z2 (X1 or Z1 or X2 or Z2), D4 (D3) cannot interact with X1 or Z1 or X2 or Z2 (X1 or Z2) at the same time. Obviously, if a two-qubit gate is being performed between two qubits, the qubits involved can not perform another single- or two-qubit gate at the same time.

As an example, in Figure 3.4, a CZ gate is applied between qubits 11 and 14 (`cz (q[11], q[14])`). Thus, both qubits need to be tuned to closer frequencies – f_1^{int} and f_2 in this case. And, as explained before, in order to not create intereferences between qubits some of them should be detunned away. Specifically, qubits 10 and 16 need to be detunned to f_2^{park} because they share a connection with the qubits performing the CZ gate. This means that no single-qubit gate can be applied to these qubits that are outside their main frequency (f_2). And, also, that they cannot perform another two-qubit gate at the same time with any other qubit from the QWG 0 – the red qubits with main frequency f_1 – because it is required to tune them to f_2.

Finally, all the qubits that are already in use cannot be used at the same time. Therefore, qubits 11 and 14 in the example cannot be used in any single- or two-qubit gates.

3.1.5. Measurement constraint

Measuring the qubits is done by using feedlines coupled to several qubits. For example, in the SC-17 chip, the feedline 1 is used to measure qubits 1, 4, 5, 7, 8, 10, 11, 14 and 15. This will lead us to the last limitation, the *measurement constraint* (hardware constraint). In this case, the measurement on a qubit cannot start when another qubit coupled to the same feedline is being measured, but it allows to start measurement on any combination of qubits coupled to the same feedline at the same time. Furthermore, there is no dependency between measurements of any two qubits coupled to different feedlines.

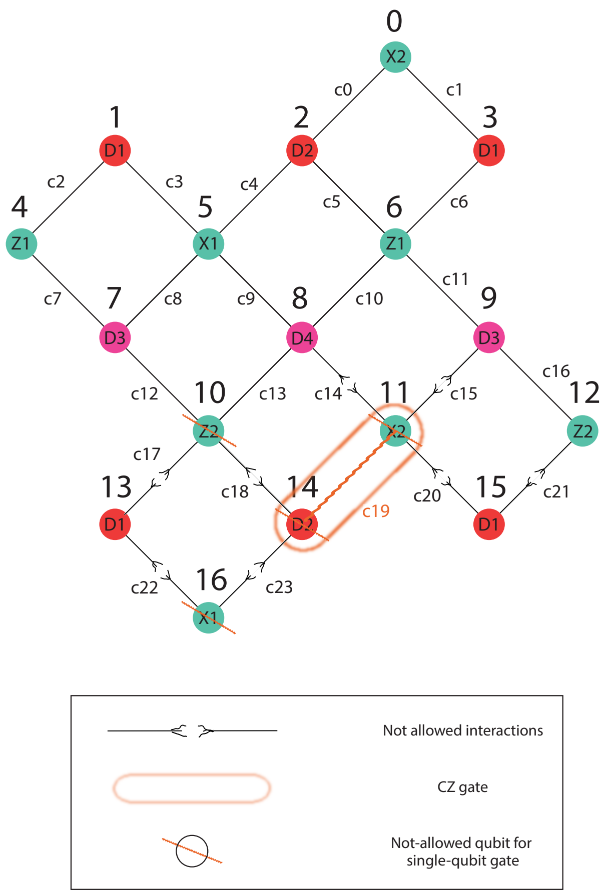


Figure 3.4: Example of a two qubit gate and allowed and not allowed parallel gates in SC-17

In SC-17, it is possible to do

```
measure q[0] | measure q[2] | measure q[3] | measure
  ↪  q[6] | measure q[9] | measure q[12]
measure q[1] | measure q[13]
qwait 30
measure q[3] | measure q[8]
```

notice that the measurement of qubits 13 and 16 is executed at the same cycle.

On the other hand it is not possible to do

```
measure q[2]
measure q[6]
```

Note that, in order to measure qubit 2 and 6 in different times, the latter instruction should wait 30 cycles for the first one to finish. In other words, this code is missing a `qwait 30` instruction.

3.2. Mapping model

In order to map quantum circuits on the superconducting quantum chip we will use the mapping model developed in our group [30]. A mapping algorithm or mapper is an algorithm able to find a mapping solution given a quantum circuit and a target device. We consider that a mapper is subdivided in three tasks: scheduler, initial placement and routing, as described in the Mapping of quantum circuits section. Our procedure is modular, although the flow is restricted to the one described below. The mapper is fully adaptable to any device constrain. As we will explain, it also offers two different kinds of schedulers and three router possibilities, as well as the option of running or not the initial placement and another options related to the chip constrains. We believe this solution will aid researchers to investigate the best mapping, with different configurations. The mapping model works as follows.

3.2.1. Initial placement

The first step is to map the virtual qubits (the ones of the circuit) to the physical ones (the ones in the quantum chip). Once the mapper knows the qubits required by the algorithm, it will try to find the best initial placement for the given circuit, that is, the placement that requires less movements of qubits. First, it will check the first set of two-qubit gates and their target qubits. If those qubits are not NN, it will initiate an Integer Liner Programming (ILP) algorithm [30] to find the optimal initial placement. This optimal initial placement tries to place the qubits in a way that the minimum number of qubit movements are required.

3.2.2. Router

The function of the router is to move non-neighbouring qubits to adjacent positions whenever they need to interact. To this purpose, our router inspects all the gates, one by one, looking for non-NN qubits interactions. Whenever the router encounters one, it will calculate several shortest paths – the ones that use the minimal amount of SWAPs – and selects one of them. This path is selected based on the circuit depth, that is, the path that better interleaves with the previous operations. Then, it will insert the SWAP operations (decomposed or not decomposed) required to follow the selected path.

There are three router options, and a different path is selected depending on the router option that was selected by the user:

1. `base` finds all the shortest paths and selects one of them randomly. The shortest paths are calculated based on the Manhattan distance, counting the number of steps that the qubits will need to move.
2. `minextend` finds all the shortest paths as well, but selects the one that will minimally increase the circuit depth. In this case, only gate dependencies are considered, not chip constraints that limits the parallelism.
3. The `minextendrc` approach is exactly the same as the second one, but taking into account the chip constraints when selecting the path.

3.2.3. Scheduler

The RC-scheduler will finally schedule the circuit using either an ALAP or an ASAP approach, depending on the user requirements. It will take into account the chip parallelism constraints, as the ones described in 3.1.

One downside factor regarding this methodology is that the mapper only takes into account one gate every time. It does not look ahead in order to decide the best path for next iterations. Or, what is the same, it does not always find the best mapping solution. This limitation comes from the fact that, the more steps you are looking ahead while mapping, the more computationally complex will the mapper be. We highlight that the mapping problem is an NP problem [22]. Therefore, this method represents a viable solution, although a lot of future work should be done.

3.3. New Mapping Metrics: algorithm's reliability

As mentioned in the Mapping metrics section in the second chapter, we name *mapping metrics* to those metrics used to assert the quality of a mapper and that are also used by the mapper algorithm as a cost function to optimize. In this section, we will present and define some new mapping metrics: **fidelity**, **probability of success** and **Quantum Volume**.

3.3.1. Fidelity and Probability of success

Fidelity and probability of success are two different ways to measure how errors affect the algorithm reliability. Fidelity is measured before measuring the qubits, and it is the difference between the obtained quantum state (affected by errors) and the expected quantum state (the one without errors). Probability of success is obtained after measuring the qubits, and then after collapsing the quantum state. Note that in this case measurement errors are also considered. After running the algorithm several times, it shows how many times

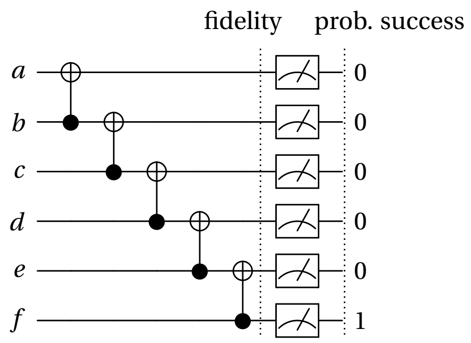


Figure 3.5: Example of the fidelity and probability of success calculation in the Gray encoder quantum circuit

a correct result is measured. Therefore, while fidelity is a theoretical metric that can only be calculated in simulations, the probability of success is a hardware metric that can be asserted in experiments in real life.

Fidelity

Fidelity compares two quantum states. Unlike the success of an algorithm, that is a boolean metric – either success or failure –, the fidelity returns a float number between 0 to 1 asserting how different the states are – how far is one state from the other. 0 fidelity would mean that the states are the same – they do not differ – and 1 would mean that states are perpendicularly different. As an analogy, if we consider the quantum states as vectors, the fidelity would be the dot product between them. However, the fidelity between orthogonal states – states that differ in rotations around the x-axis as $|0100\rangle$, $|0001\rangle$ and $|1111\rangle$ – is also 0. For more information, we refer to [3].

Probability of success

We define probability of success as the probability of having the correct or expected results after running and measuring several times a quantum algorithm. For instance, if the expected result of an algorithm is `0100` and the results are either `0001` or `1111` we will get a failure. We will only get a success if the algorithm returns `0100`. Then, in the previous example, if we get `0001`, `0100` and `1111` as results after running the algorithm three times the resulting probability of success will be $\frac{1}{3}$.

In our study, we will use both metrics in order to study how the mapping process and then the amount of errors affect the algorithm reliability. Note that as we already mention, the fidelity is calculated before measurement and then it does not take into account the measurement errors. It could be the case that a quantum state is erroneous before the measurement and, then, due to the error added from the measurement the erroneous state would flip and converge into the expected state. At the same time, a correct quantum state could be measured wrongly due to the measurement errors.

It is worth noting that both metrics, the fidelity and probability of success, are obtained by using a quantum computer simulator as we will explain in the next chapter.

3.3.2. Quantum Volume

Another metric that could be useful for the mapping quality assessment is Quantum Volume [36, 37]. Quantum Volume is a metric proposed by IBM to depict whether a device is able to run a quantum circuit or not. Given the different hardware implementations and technologies in Quantum Computation (superconducting, ion-trap, spin qubits, ...), it is often difficult to benchmark the usefulness or power of quantum systems. The aim of Quantum Volume is to quantify the computational power of quantum devices. Consequently we will use it as a metric to measure the runnability of the quantum algorithms on the quantum devices. But, while the device is the target of the Quantum Volume metric, we focus on the circuit. Our goal is to assess how the mapping procedure affects the runnability of a given circuit and to study how the Quantum Volume is related to the fidelity and the probability of success.

The general Quantum Volume formula is defined in eq. 3.1 where N is the number of physical qubits and $d(N)$ is the achievable circuit depth, i.e. the maximum circuit depth for which the results, after running it on some device, are correctable and useful. $d(N)$ depends on the number of qubits in the circuit as well as on the error rate of the quantum device.

$$V_Q = \min(N, d(N))^2 \tag{3.1}$$

In our case, we want to relate the Quantum Volume of a device with the circuit. Therefore, we define the **algorithms Quantum Volume** and the **runnability** of the quantum circuit on a given device.

As with V_Q, we initially derived the algorithm's Quantum Volume from the general equation V_Q (see eq. 3.2), although we will adapt it later.

$$V_Q^a = \min[n, d]^2 \tag{3.2}$$

Note that d is not $d(N)$ but the real depth of the given algorithm. At the same time, n is the number of qubits required by the algorithm itself.

We are aware that this approach has a limitation regarding the mapping of the quantum circuit. As explained before, V_Q is able to take into account the sophistication of the mapping procedure. It is inherited in the *model algorithm*. But, in this case, the V_Q^a of an algorithm before and after mapping will remain the same. After mapping an algorithm, the usual effect is an increase in the depth and the number of operations. Rare mapping methods consider the qubit addition in the technique. And, even considering it, n is not often growing too much in comparison with d. In the current NISQ era, the quantum circuits need much less qubits than depth. Therefore, most of the times, the minimum value between n and d will be n. As soon as V_Q^a is taking into account the minimum of them and the mapping procedure affects mostly to d we can conclude that this definition of V_Q^a is not considering the mapping in its results.

A simplified solution for this problem would be the V_Q^a definition as the multiplication between n and d (see eq. 3.3). Unfortunately, this approach has several drawbacks as well. As Moll et al. point out [36], extreme cases of high n and low d – or the other way around – lead to inconsistencies of the multiplication metric. But, considering that most of our work is not going to be in any of these extreme cases and that we can avoid those outliers, we define the algorithm's Quantum Volume as:

$$V_Q^a = n \times d \tag{3.3}$$

Finally, once the Quantum Volume of an algorithm is stated, we define runnability as the condition for which the V_Q should be bigger than V_Q^a. That is the condition that the computational power of the device should be bigger than the computational power required by the algorithm.

$$\text{Runnable if: } V_Q > V_Q^a \qquad \text{when } N \geq n \tag{3.4}$$

For instance, in order to understand this concept, one may imagine the process of checking, whether or not, some cube with a given volume – representing the algorithm – would fit in a box – the device –. If the algorithm's box volume is smaller than the volume of the device's box, the algorithm's box will fit inside.

Indeed, one acceptable criticism of this definition is that, as V_Q and V_Q^a are finally defined in the previous sections, it seems that it is not really fair to compare them. But, as soon as the general behaviour of both definitions of V_Q^a is the same we believe that this definition of runnability is mathematically correct and useful.

4 Benchmarks and analysis framework

In this chapter we describe the benchmarks that will be mapped to the Surface-17 chip and the simulation framework develop for the study of the mapping metrics.

4.1. Benchmarks

In order to analyze the different metrics to assess the quality of a mapping algorithm, we selected a set of quantum algorithms available in the literature.

I built a list of quantum algorithms coming from different sources. The sources we chose are RevLib [38], ScaffCC [39], some benchmarks from Zulehner's et al. work [21] and QLib [40] because of the variety of algorithms and because they are already decomposed in the Clifford+T set. Note that the language used to describe the algorithm varies from source to source. Although, they all use a QASM-based language, there are small syntax differences. As we will explain in the next section, we will use our own programming languages – OpenQL and cQASM – and compiler. Therefore we had to translate all the selected algorithms from their QASM versions to the OpenQL notation. I developed a parser in order to do that.

I also profiled each benchmark including: the number of qubits of the algorithm, the number of gates, the functionality of the algorithm and different graphs, showing the percentage of the operations types or the degree of parallelism of the algorithm. Most of the benchmarks are classical algorithms adapted to quantum circuits. They are deterministic and then the result is always the same if there are no errors. This is because at the end of the circuit, before measuring the qubits, qubits are in either ground or excited state, but never superposition.

The benchmarks were also classified by their functionality. This is an important step because algorithms that do similar calculations use to have common gates distribution. From all the benchmarks we have, we can distinguish six different classes.

- Quantum Gates: Circuits that are a decomposition of a Quantum Gate
- Search Algorithms
- Worst Cases: Circuits that were really difficult to generate for RevLib
  - HWB: is the simplest function with exponential Ordered Binary Decision Diagrams (OBDD) size.
- Encoding Functions: Classical codification functions
- Arithmetic Functions: Functions that perform an arithmetic operation
- Miscellaneous: Mix of different kind of algorithms

We came up with a list of 84 different algorithms – 697 benchmarks taking into account the different versions of the same functionality – that can be found in the qbench Github repo, as well as the previous information in much more detail. These algorithms cannot only be used for the mapping problem but also for other activities in our group.

4.1.1. Benchmark selection

In order to have a small, but representative, set of benchmarks to map, we selected just some of them. This requirement comes by the fact that simulations are long and computationally exhaustive. In order to do that, we studied the different benchmark profiles looking for most illustrative cases. The criteria used for selecting the algorithms used in this work are: i) the number of qubits of the algorithm. It should be less than 17 as we will use the Surface-17 chip; ii) the number of gates. We select benchmarks with a number of gates as spread as possible; iii) different functionality. We try to have the least number of versions of the same algorithm as possible. We only take the same algorithm versions for cases in which the number of qubits and the number of gates show an interesting combination. For example, two similar algorithms in which number of qubits or number of gates are very distinct appear as the first three selected benchmarks in Tab. 5.1.

Once we had our requirements we could start the analysis and the selection afterwards. In Tab. 5.1 we show the final benchmark selection. 43 benchmarks (with qubits numbers from 3 to 17 qubits) were selected after

Table 4.1: Restriction summary of the benchmark selection

Criteria: - # qubits < 17 - # gates as spread as possible and in the case of repeated benchmark the minimum number of gates - The less number of the same algorithm versions/classes as possible - The benchmarks that are repeated and have an interesting combination of No. qubits/No. gates are preferred

applying the previous restrictions to the analysis of the benchmarks described in the next section (see Table 5.1). After simulating the algorithms, few of them returned errors (segmentation fault) due to the high computational level they required and some others did not finish the simulation in less than two weeks; so we discarded them.

4.2. Analysis framework

In this section we introduce the framework used to map the quantum algorithms and analyze the quantum metrics. It is based on two steps: **compilation** and **simulation**. With this purpose we connect two tools, OpenQL [41] and quantumsim [4], and we build a whole framework over them.

4.2.1. Compiler (OpenQL)

OpenQL is a framework for high-level quantum programming developed by our group to describe quantum algorithms and compile them. The framework can be found as a library in either C++ or Python and, therefore, the circuits are described over one of these programming languages. It can target real quantum chips as well as different quantum computer simulators such are QX simulator [42] or quantumsim [4].

One of the passes of the compiler is the **mapper** algorithm developed by our group. It performs the initial placement, schedule the operations and route the qubits taking into account the quantum chip constraints that are described in a JSON file. It is also able to load different compiler configurations like the kind of scheduler, router or initial placement. We will use OpenQL to describe the selected benchmarks and compile them for the SC-17 chip. In Fig. 4.1 we show an example of OpenQL code using Python that describes the Gray encoder quantum circuit (Fig. 2.2). More insights can be found in the github repository.

```python
from openql import openql as ql

def circuit(config_file, scheduler='ASAP', uniform_sched= 'no', mapper='base', initial_placement='no',
  ↪   output_dir_name='test_output', optimize='no', measurement=True, log_level='LOG_WARNING'):
    curdir = os.path.dirname(__file__)
    output_dir = os.path.join(curdir, output_dir_name)
    ql.set_option('output_dir', output_dir)
    ql.set_option('optimize', optimize)
    ql.set_option('scheduler', scheduler)
    ql.set_option('scheduler_uniform', uniform_sched)
    ql.set_option('mapper', mapper)
    ql.set_option('initialplace', initial_placement)
    ql.set_option('log_level', log_level)

    config_fn = os.path.join(curdir, config_file)

    platform  = ql.Platform('starmon', config_fn)
    sweep_points = [1,2]
    num_circuits = 1
    num_qubits = 6
    p = ql.Program('graycode6', platform, num_qubits)
    p.set_sweep_points(sweep_points, num_circuits)
    k = ql.Kernel('graycode6', platform, num_qubits)
    k.gate('cnot',[1,0])
    k.gate('cnot',[2,1])
    k.gate('cnot',[3,2])
    k.gate('cnot',[4,3])
    k.gate('cnot',[5,4])

    if measurement:
        for q in range(num_qubits):
            k.gate('measure', [q])

    p.add_kernel(k)
    p.compile()
```

Figure 4.1: OpenQL description in python code describing the Gray code algorithm.

4.2.2. quantumsim

Quantumsim [4] is a simulator for superconducting systems and designed to study the SC-17 chip. Its error model has been induced from the chip's behaviour after several experiments. Therefore, quantumsim's error model is much more complete for the superconducting case than other general simulators. It is based on the decoherence time and the observed gate error rates. Although the detail in its error model makes each simulation computationally harder. As a matter of fact, quantumsim is able to boost its calculations with a Graphics Processing Unit (GPU). Quantumsim can be found as a **python library** in its github repository with instructions to install it and an overview of how to use it.

4.2.3. Analysis Framework

In order to study the metrics, we developed a simulation framework. This framework was developed in order to be a tool extensively used by our group. For this reason it was thought as a modular system. The framework maps a quantum algorithm to a given device, using OpenQL. Then it simulates it with quantumsim. Later on, the framework will execute an analysis of the simulation results. The analysis is based on the fidelity, probability of success and quantum volume calculations besides the depth, number of added SWAPs and the total number of operations extracted from the different circuit descriptions. Finally, all the data is stored in a database with the aim to do an exercise of data analysis to study the mapping metrics. The framework flow can be followed in the Fig. 4.2. Note that, although we choose quantumsim to run our experiments, we could have introduced any other simulator due to the modular nature of the framework.

1. Benchmark mapping

   First, the framework compiles the benchmarks described in OpenQL depending on the **configuration** introduced. The configuration options are the mapper characteristics (scheduler, initial placement and router) and the JSON file describing the quantum device for which the algorithm should be mapped. OpenQL exports the mapped version of the circuit in a language understandable by a simulator, in either quantumsim or cQASM [43] code. From all the circuits descriptions – before and after being mapped –, the framework extracts the algorithm name, the number of qubits used and the number of operations. Also, from the circuits before mapping it will extract the source and the functionality and from the ones mapped the depth and the number of SWAPs added. All this information will be stored in the database.

2. Mapping simulation

   After being mapped, the simulator will load the mapped circuit and the simulation characteristics. The mapped algorithm will be simulated first without errors, in order to know what is the correct result or the result we should expect from this circuit. We store both, the resultant quantum state ψ and its ideal measurement $M(\psi)$. Note that, as long as our benchmarks are all deterministic (see the Benchmarks section) the quantum state and the measurement will be the same. After saving the correct result, the framework proceeds to simulate N times the benchmark. The results affected by the errors will be used to calculate the fidelity – between the quantum state $\psi^{'}$ and the expected one ψ – as well as the probability of success – between the measurement $M(\psi^{'})$ and the expected one $M(\psi)$. The final value for fidelity and probability of success will be calculated averaging all the fidelities and probabilities of success calculated per simulation. At the same time the quantum volume will be calculated with the depth and the number of qubits from the qubits mapped. Finally these results and the simulations parameters will be stored in the database.

3. Database

   The database is conformed by six different tables. The *Benchmarks* table will store the information from the circuits before being mapped: the algorithm name, its source and functionality, the number of qubits used and the number of operations. The *Configurations* table will save the device configuration and the mapper characteristics. *HardwareBenchs* will store the name, the number of qubits and operations from the mapped algorithms as well as the SWAPs added and the circuit depth. *Simulations* will save the simulator options as the simulator used, the number of simulations, the error rate, the decoherence times and the measurement error. The *Results* table stores all the results from the simulations and the *Experiments* table saves data about the moment when the framework was used in order to identify the different experiments done.

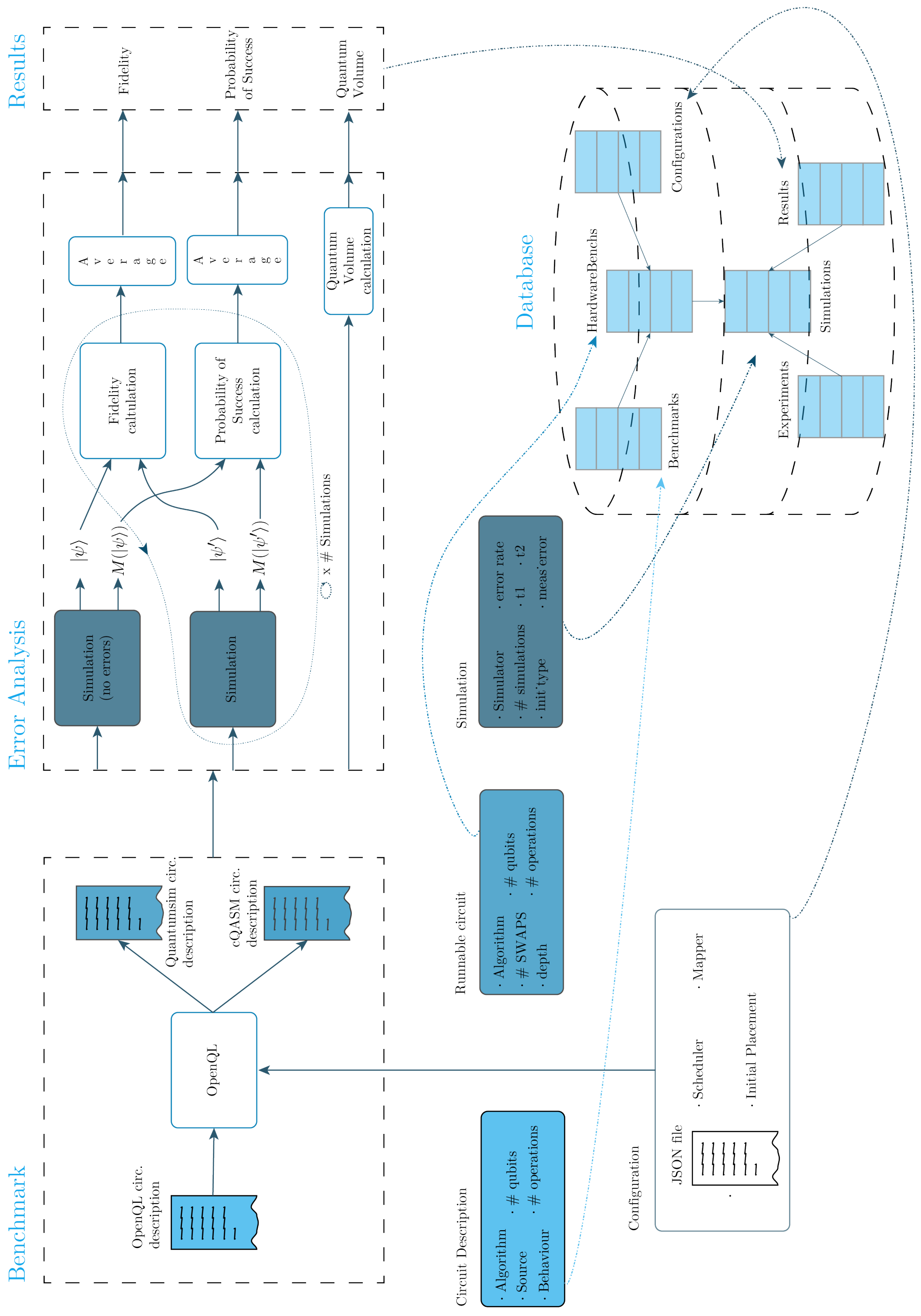


Figure 4.2: Analysis Framework

5
Results

In this chapter the simulation results are presented with reference to the aim of the thesis, which was to study the mapping metrics: their precision when assessing the mapping quality and their relation with the error amount after mapping. First, in the Impact of the mapping on the algorithm reliability section we show how the mapping procedure affects the general increment of errors. After that, we analyze the metrics and their correlation with the error increase in the Analysis of the mapping metrics section.

5.1. Impact of the mapping on the algorithm reliability

After the selection of benchmarks in the Benchmarks section, we started their simulations using our simulation framework. Some of the benchmarks either had very long simulation times or were even impossible to simulate due the computational power required by the quantumsim's complex error model. In most of those cases the circuits had more than ten qubits. As we mentioned before throughout this thesis, the simulation of quantum systems is computationally exhausting. The higher the number of qubits or the length of the circuit, the harder is it to simulate it. Indeed, it is a critical issue in our case, as soon as we need to run multiple simulations in a complex error model. Therefore, as can be seen in Tab. 5.1, we address that the final benchmark selection has a limitation in the number of qubits.

As explained in the Analysis framework section, after running each benchmark for one thousand times, the results obtained are the fidelity and the probability of success. We also extracted other metrics like the number of SWAPs added, the depth of the circuits and the Quantum Volume, among other circuit statistics. These metrics were obtained for all mentioned benchmarks before (non-mapped) and after mapping (mapped). Note that, for a fair comparison both the mapped and the non-mapped circuits have been decomposed into the gates supported by the Surface-17 chip (see Fig. 3.2). For the mapped ones, we use the three router algorithms developed in our group (see Mapping Model): *base*, *minextend* and *minextendrc*.

In quantumsim, we also try different configurations regarding the decoherence time in order to study the mappers in different error regimes. All the results are detailed in Appendix A.

In order to illustrate how the mapping process affects the algorithm reliability, Figure 5.1 shows the fidelity for some of the benchmarks before being mapped and after being mapped using the `minextendrc` router and a decoherence time of 30μs. We can see that fidelity is smaller for long circuits – like `sf_274` or `mod5adder_127` – than for the short ones – `graycode6_47` or `xor5_254`. It can be seen that, for the long circuits, the fidelity can even decrease more than 50%. For instance, `sf_274`'s fidelity goes from 0.35 to 0.17, `mod5adder_127`'s goes from 0.45 to 0.19. On the other hand, for the small circuits like `graycode6_47` or `xor5_254` the fidelity decreases around 1%; from 0.99 to 0.98 and from 0.99 to 0.97, respectively. For more details, we show the exact result values for all the benchmarks in Appendix A.

As we mentioned before, we use three different routers, each one giving a different mapped version per benchmark and, therefore, different metric statistics. For instance, as it can be seen in Appendix A and in

Table 5.1: Table of the selected benchmarks to be mapped.

Benchmark	# qubits	# gates	two-qubit gates (fraction)
graycode6_47	6	5	1.000
xor5_254	6	7	0.714
4mod5_v0_20	5	20	0.500
ham3_102	3	20	0.550
mod5d1_63	5	22	0.591
4gt11_82	5	27	0.667
rd32_v0_66	4	34	0.471
alu_v0_27	5	36	0.472
4mod5_bdd_287	7	70	0.443
one_two_three_v3_101	5	70	0.457
decod24_bdd_294	6	73	0.438
alu_bdd_288	7	84	0.452
one_two_three_v1_99	5	132	0.447
mod10_176	5	178	0.438
4gt12_v1_89	6	228	0.439
hwb4_49	5	233	0.459
4gt4_v0_72	6	258	0.438
decod24_enable_126	6	338	0.441
mod8_10_177	6	440	0.445
mod5adder_127	6	555	0.431
sf_276	6	778	0.432
sf_274	6	781	0.430
sym6_145	7	3888	0.438

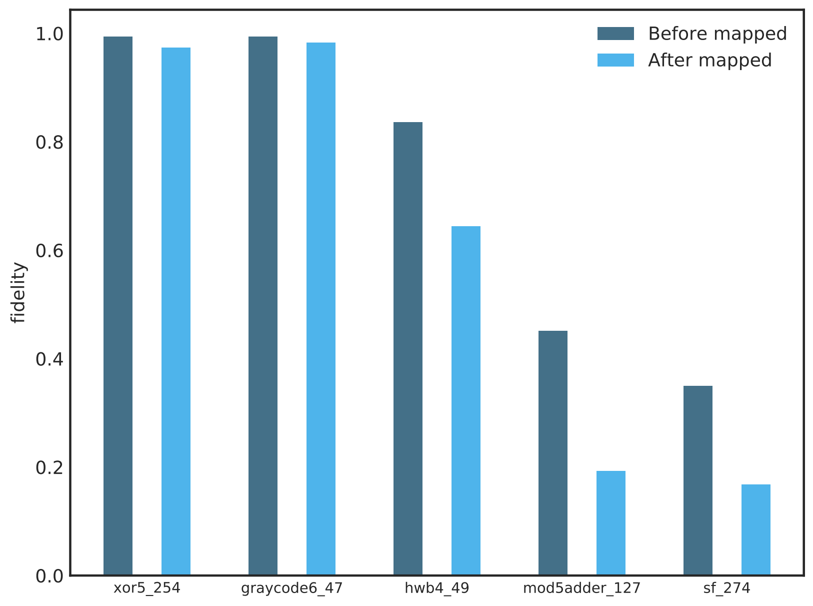


Figure 5.1: Difference of fidelities before and after mapping with the *minextendrc* router for five different benchmarks.

Tab. 5.2, the depth of the graycode6_47 circuit before being mapped is 32 cycles and after being mapped grows to 111, 61 or 82 depending on the router.

In Fig. 5.2(a) and Fig. 5.3(a) we plot the fidelity of some of the benchmarks non-mapped and mapped, using the three different versions of the router. Benchmarks are ordered from the shortest to the longest circuit depth, from left to right, as shown in Tab. 5.2. The dark blue dots represent the benchmarks before being mapped and the light blue ones represent the different mapped version of it from the different routers. Fig. 5.2(a) presents the result with a decoherence time of 30μs and Fig. 5.3(a) the ones with a decoherence time of 10μs. As mentioned in the Qubits are faulty section, the shorter the decoherence time the more errors would appear. From these results, we observe that, in general: i) the fidelity decreases as the circuit depth increases, ii) the shorter the decoherence time (t_d) the lower the fidelity and the faster it decreases and iii) as we showed before, the fidelity decreases after the mapping. This behaviour is certainly because the longer the circuit, the more errors it will get.

To analyze, the difference in fidelity between the mapped and non-mapped cicuits, in Fig. 5.2(b) and Fig. 5.3(b) we plot the maximum (red) and minimum (blue) percentage of difference in fidelity. We calculate the percentage of the fidelity difference as $\frac{f_{\text{before}}-f_{\text{after}}}{f_{\text{before}}}$ where f_{after} is the fidelity of the mapped circuit and f_{before} is the fidelity of the non-mapped one. In Fig. 5.2(a) and Fig. 5.3(a), this difference is illustrated as a red and a blue lines.

We can observe how the fidelity difference between non-mapped and the mapped algorithms tends to grow. As soon as most of the selected benchmarks have a similar two-qubit gates percentage and also a similar number of qubits – main parameters for the mapping task –, we can say that this increasing tendency is mostly is due to the length of the circuit before being mapped. In other words, the longest the circuit, the higher the difference between fidelities. Moreover, another remark we see is that, in the case of a decoherence time of 10μs (Fig 5.3), we observe strange results like negative fidelities. This is due to the chaotic behaviour of the quantum system whenever the latency of the circuit is very close or longer than the decoherence time of the qubit. As we explained in the Qubits are faulty section, the decoherence time is related with the amount of errors that affect our circuit; the shorter the decoherence time, the more errors will appear in our quantum system.

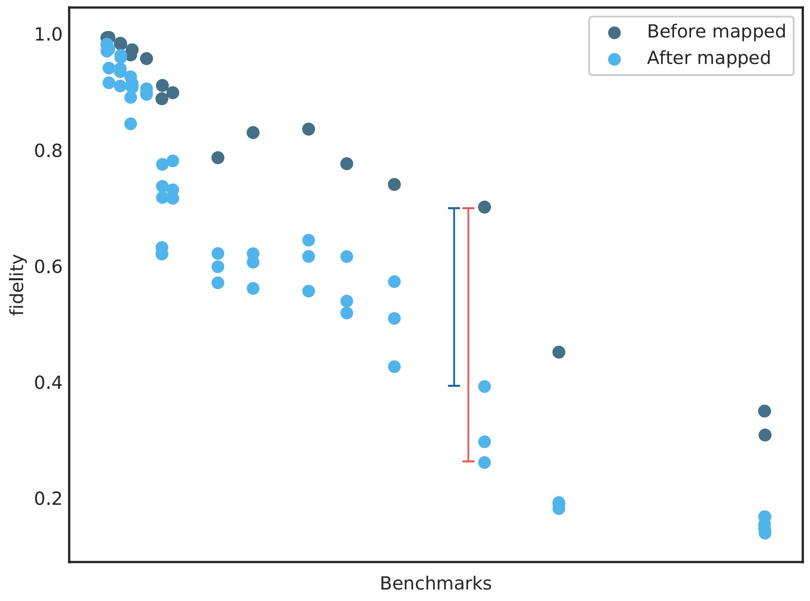
Table 5.2: Different depth per benchmark

Benchmark	Depth before mapping	Depth after mapping with *minextendrc*	Depth after mapping with *minextend*	Depth after mapping with *base*
graycode6_47	32	111	61	82
mod5d1_63	59	209	136	146
ham3_102	60	127	121	98
alu_v0_27	80	248	156	214
miller_11	112	307	278	231
one_two_three_v3_101	143	440	302	323
decod24_bdd_294	144	407	328	300
alu_bdd_288	165	495	383	360
one_two_three_v1_99	256	839	530	609
mod10_176	327	1090	687	734
hwb4_49	439	1387	961	1006
mini_alu_167	516	1598	992	1274
decod24_enable_126	612	1788	1440	1446
mod8_10_177	794	2275	1761	2006
mod5adder_127	944	2878	2667	2378

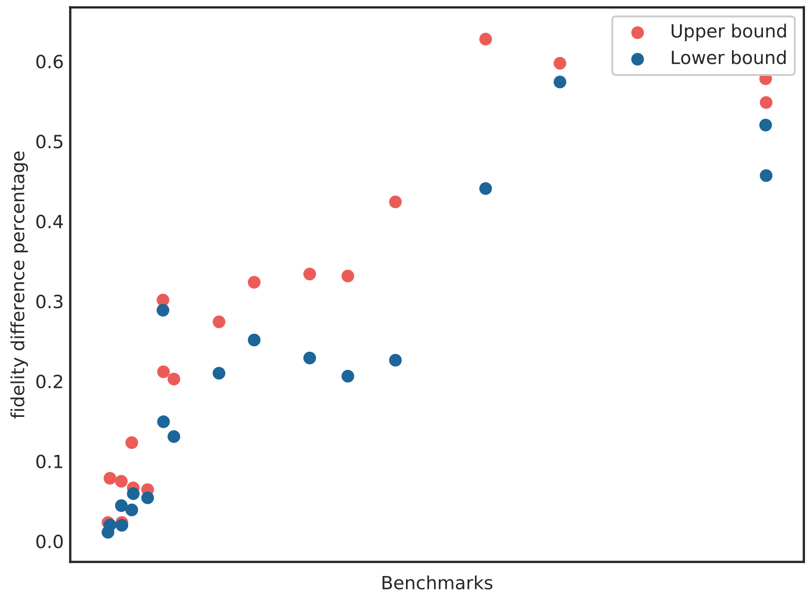
5.2. Analysis of the mapping metrics

Once we acknowledge the behaviour of the fidelity decrease due to the mapping process, in this section, we will analyze how different circuit parameters affect the fidelity and the probability of success. If we are able to answer which metric is more correlated with the appearance of errors in the circuit, we will know the most critical metric for the mapping quality. Therefore, in this section we evaluate the behaviour of the number of gates, the number of two-qubit gates – in accordance with the number of SWAPs –, the depth and the Quantum Volume metrics in comparison with the algorithm's reliability, or what is the same, the fidelity and the probability of success.

Before doing that, first we analyze the probability of success and fidelity correlation. As we explained in the Fidelity and Probability of success section, one of the main differences between these two metrics is that the probability of success takes into account the measurement process of the qubits, while fidelity does not. This makes the probability of success not only sensitive to the errors related with the measurement but also to the inherent non-deterministic behaviour of it. This means that, although our benchmarks are

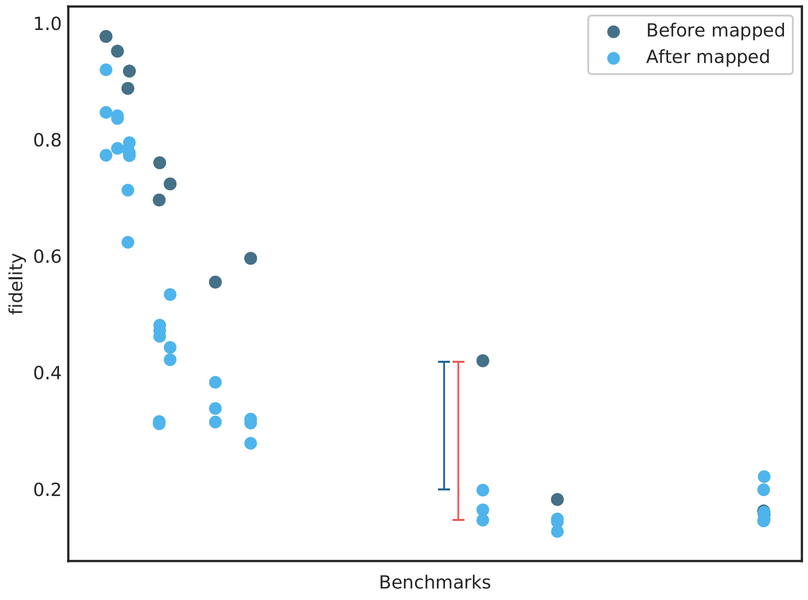


(a) Fidelity per benchmark

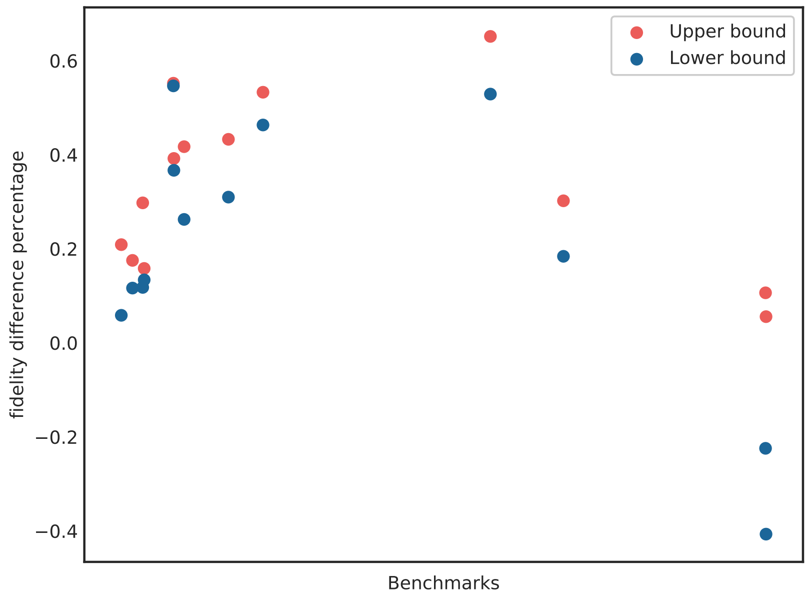


(b) Difference of fidelity per benchmark

Figure 5.2: Impact of mapping for $t_d = 30\mu s$



(a) Fidelity per benchmark



(b) Difference of fidelity per benchmark

Figure 5.3: Impact of mapping for $t_d = 10\mu s$

deterministic, a circuit could result erroneously in a state in superposition due to the quantum noise, forcing the measurement to collapse to either the correct or the wrong state. For example, let us say that we are expecting 1 as the result of a given qubit but, before measuring instead of $|1\rangle$ we got $\sqrt{0.3}|0\rangle + \sqrt{0.7}|1\rangle$. In this case, without taking into account the measurement errors, the measurement will result in 0 with a 30% chance and 1 with a 70% chance. For this reason, the probability of success tends to be more chaotic in regimes with high amount of errors; as we will throughout this section.

In Fig. 5.4 we plot the correlation of probability of success and fidelity. For this figure and the figures from now on in this section, each dot represents a different benchmark, before being mapped and also after being mapped with the three router versions (see Appendix A). The colors represent different decoherence times, blue for 30 μs and orange for 10 μs. Note that for the rest of the results, we are not interested in analyzing the difference between mapped and non-mapped algorithms (as it has been already analyzed in previous section). In this case we just treat them as circuits with different number of gates or circuit depth, or Quantum Volume.

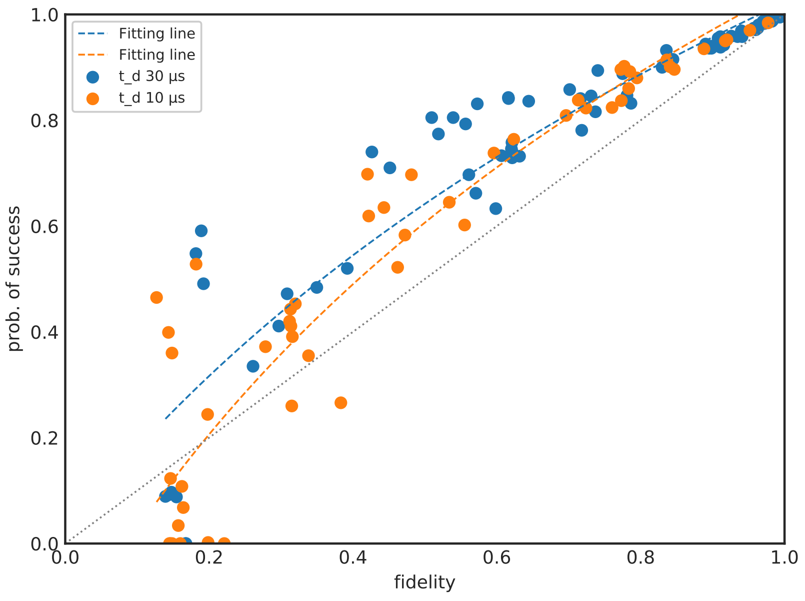


Figure 5.4: Correlation between fidelity and probability of success for two different decoherence times

As expected, our experiments prove that both metrics are highly correlated. We also appreciated the fact that most of the samples are above the $x = y$ line; meaning that the probability of success, in general, is higher than the fidelity. This could suggest that the measurement is 'correcting' circuit errors collapsing the state in the correct result, instead of the wrong one. It seems that the measurement is committing 'good' mistakes that result in the expected solution. That is, this behaviour could be happening because of the indeterministic behaviour of the measurement. Further research should be done to better understand this behaviour.

Another observation we see is that, the closer the samples get to 0 fidelity or 0 probability of success, the more chaotic the values are. Actually we can delimit two parts in the graph: above and below 0.7 in fidelity – around 0.8 in probability of success. Above 0.7 we observe an almost linear behaviour and below 0.6 we start seeing more chaotic results. This behaviour happens because, whenever a circuit accumulates a high amount of errors, fidelity or probability of success low but almost random values. Finally, we see that most of the orange samples are below the 0.6 fidelity value; or what is the same, most of the benchmarks simulated with the low decoherence time have more chaotic values. This can be explained by the fact that the correctness of circuits that are run in lower error rate regimes will have much better results in systems with higher error rates.

Now we will analyze the relation between the fidelity and the probability of succes with the number of gates, number of two-qubit gates, circuit depth and Quantum Volume (new metric).

The results of the main mapping metrics against fidelity are depicted in Fig. 5.5. We observe that, for all the cases, the fidelity decreases with a behaviour that looks inverse exponential and that decreases faster for small decoherence times. Certainly, the shorter the decoherence times we use, the more benchmarks will have non-useful results. We can also see how fidelity never goes to zero, but it gets constant around 0.2, due to the randomness of the results. We consider the point where fidelity is constant as the limit in terms of each one of the variables. We plot a line for the orange samples to mark this point. Note that in the case of the blue samples this point is not visible. Finally, it can be seen how the number of gates is the metric most related with the fidelity; it is the one with the samples less scattered.

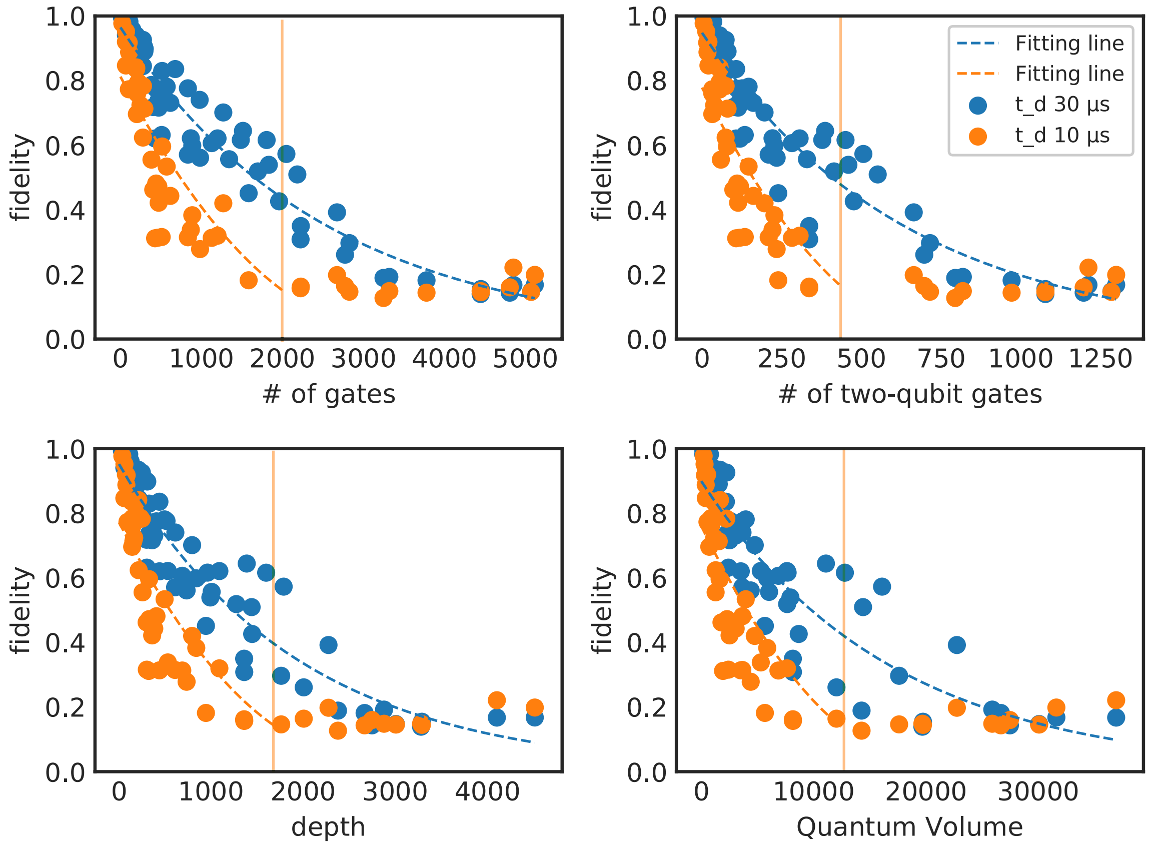


Figure 5.5: Correlation between fidelity and the mapping metrics.

The correlation between the probability of success and the other metrics can be see in Fig. 5.6. We observe a decreasing behaviour, although the samples are much more spread in this case than in the case of the fidelity. We believe that this is caused by the indeterministic effect in the nature of the measurement. We can see again that for shorter decoherence times the probability of success is lower and descends much faster.

In general, we observe a certain correlation between all of the metrics and the fidelity and the probability of success; so in order to differ between the metrics, we calculate the Pearson correlation coefficients to measure the correlation quality between them for different amounts of errors in a circuit. Note that the Pearson coefficient measures linear correlations and, as it looks like in Fig. 5.5, the metrics behave in an inverse exponential fashion against fidelity. For this reason, in order to see the real correlation between the metrics and the fidelity, we applied a $log(\cdot)$ transformation to our fidelity data in order to make it linear for the Pearson calculation.

As it can be seen in the Pearson values (Tab. 5.3 and Tab. 5.4) the most correlated metric is the number of gates. Then, this suggests that for our specific case, the mapping algorithm should optimize in terms of number of gates, or what is the same, in number of SWAPs. As Tab. 5.3 and Tab. 5.4 highlight, we have a worse correlation for the shorter decoherence time. This lack of correlation can be attributed to the fact that the majority of the samples with $t_d = 1000$ are highly affected by the errors and, therefore, the samples have more random values.

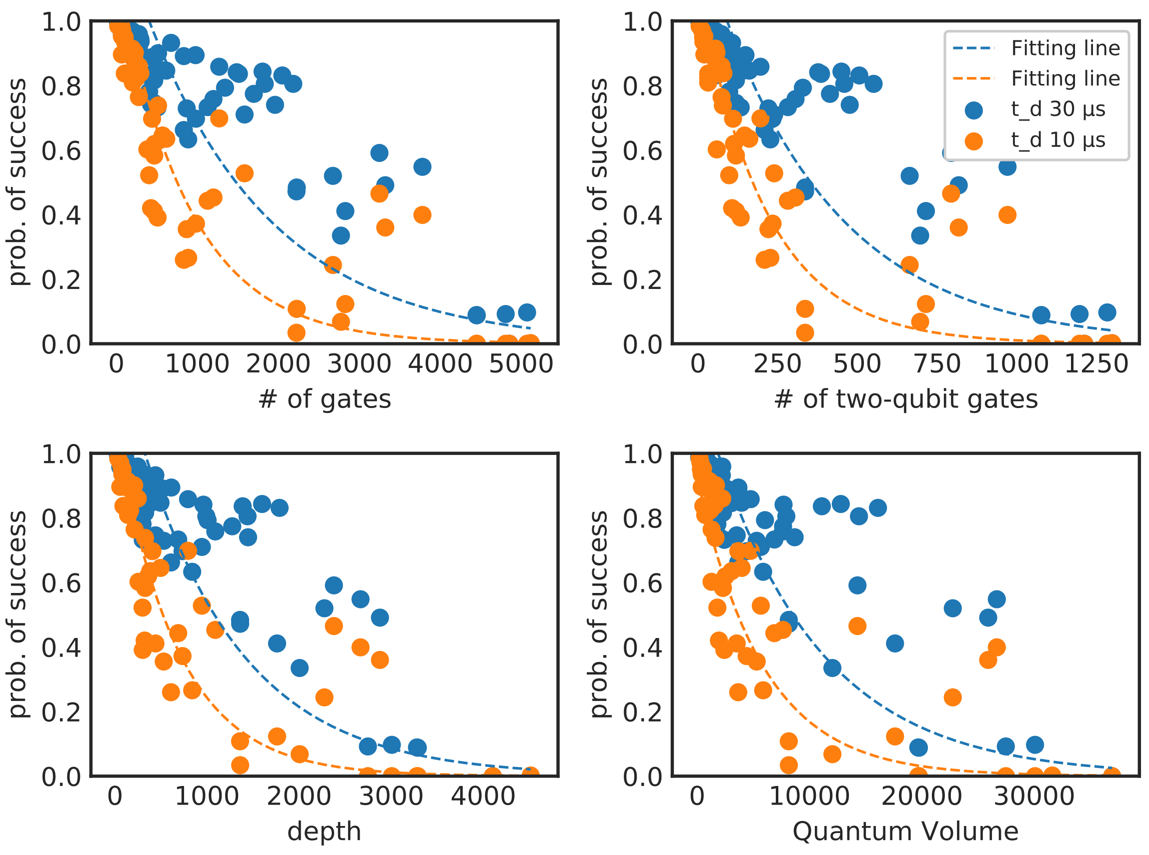


Figure 5.6: Correlation between probability of success and the mapping metrics.

Moreover, contrary to our expectations, the Quantum Volume is the least correlated metric. This small lack of correlation can be attributed to the imprecise formula that we chose to compute it. Future work needs to be done to devise a better formula. Note that the idea of introducing Quantum Volume as a metric was to have an easier way to calculate the algorithm's reliability without having to obtain the probability of success and the fidelity through simulations.

Finally, if we compare both tables, we can see that the fidelity is more correlated with the metrics than the probability of success. It is very likely that the reason for this result is that the measurement error together with the measurement indeterministic behaviour adds more noise and spreads our samples. Also the 'correcting-errors behaviour' of the measurement should be taken into account.

Table 5.3: Pearson correlation coefficient of the log transformation of fidelity against the metrics($\rho_{log(f),Y}$), where Y is one of the four metrics we analyze

	# of Gates	# of Two-qubit gates	Depth	V_Q
$t_d = 3000$	-0.9730	-0.9600	-0.9455	-0.9118
$t_d = 1000$	-0.8466	-0.8135	-0.8093	-0.7736

Table 5.4: Pearson correlation coefficient for the probability of success against the metrics ($\rho_{p_s,Y}$), where Y is one of the four metrics we analyze

	# of Gates	# of Two-qubit gates	Depth	V_Q
$t_d = 3000$	-0.9363	-0.9248	-0.9179	-0.8797
$t_d = 1000$	-0.8341	-0.8097	-0.8076	-0.7686

6 Conclusions and Future Work

6.1. Conclusions

In this thesis we have proposed three different metrics to analyze how the mapping process affects the reliability of the algorithms. These metrics are the quantum fidelity, the probability of success of the algorithms and the Quantum Volume. They can be used not only to evaluate how good the mapping model is, but also as parameters to optimize during the mapping. Note that to calculate the fidelity and the probability of success of a quantum algorithm we need to simulate it – using a quantum simulator – which is time consuming and limits you to small circuit sizes. That is why we have investigated the Quantum Volume as possible alternative.

To this purpose we have developed a simulation framework combining the OpenQL mapping model, that takes into account the constrains of a real chip – the SC-17 –, and the quantumsim simulator. The analysis framework also stores the results of the simulations as well as the results from the mapping model in a database.

The results of this study indicates that, as expected, mapping quantum algorithm on a quantum processor decreases the algorithm's reliability (quantum fidelity). And this difference in fidelity is more pronounced for longer circuits.

In addition, although correlated, there is a slight difference between probability of success and fidelity due to the non-deterministic behaviour of the quantum measurements.

We have also investigated the correlation of both metrics, the fidelity and probability of success, with the number of gates, number of two-qubit gates, circuit depth and Quantum Volume. Our results suggest that the number of gates is the most correlated metric and then, the best candidate to optimize by the mapping model, for this specific case.

Finally, although we observe a certain correlation between fidelity/probability of success with Quantum Volume, the use of Quantum Volume to measure the algorithm's reliability should be further investigated.

6.2. Future work

While reviewing our results we identify several points that require further research.

Bigger selection of benchmarks

Although our results explore a big set of values for each metric, we are aware that some regions in our plots are much more populated in terms of samples than others. In order to fully understand the behaviour of the metrics, more and different benchmarks should be analyzed with our simulations framework. Therefore, future work should include a bigger benchmark selection.

More qubits

Also, we targeted a similar benchmarks distribution in terms of number of qubits. Due to quantumsim limitations, the number of qubits are between 3 and 7 qubits, which is a low amount of qubits. Therefore, in order to explore higher systems in terms of qubits, we suggest that further research should be undertaken in the improvement of simulators. If we are able to simulate higher amounts of qubits, we will be able to study the behaviour of the mapping and the metrics in higher qubits systems. To this purpose we could use the QX simulator, developed at the QCA lab, that will include more realistic error models and will allow the exploration of quantum systems with higher number of qubits.

Improve Quantum Volume

As we mention in the Quantum Volume section, the metric of Quantum Volume defined as the multiplication of qubits and depth of the circuit is an initial attempt of studying it as mapping metric. As we showed in the Results chapter, this definition, although correlated with the fidelity/probability of success, it is not having the best results. Thus, further studies, which take a better definition of Quantum Volume, will need to be undertaken.

Fidelity and probability of success estimation

One of the main restrictions we have found through the development of this work are the simulation limits. Simulations of long circuits require long time to be simulated and, in some cases, are not even able to be simulated. In order to avoid this overload that drags out our work we would like to model fidelity and probability of success given the different circuit parameters. Further studies on different and more samples configurations are therefore required in order to elucidate the regression model able to estimate fidelity and probability of success with the least error.

Understanding more about probability of success

We observed in the results chapter that probability of success tends to be higher than the fidelity. We are not sure about the source of this behaviour, therefore further investigation is needed.

Appendix A

In this appendix we add the exact results of the simulations for each one of the benchmarks and each of the configurations we used. Note that the non-mapped version of the benchmarks have the quantum gates decomposed in the operations allowed in the SC-17 chip. Also, notice that not all the benchmarks have given results for the simulations with a decoherence time of 10μs due to the long simulation times.

4gt11_82

Table 1: Step 1 results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	depth	# gates	# SWAPS	p. success	f	V_Q
no	5	78	84	0	0.96	0.97823066	390
minextendrc	7	226	237	17	0.929	0.92937318	1582
minextend	8	**158**	**228**	**16**	**0.947**	**0.9312172**	1264
base	6	177	**228**	**16**	0.932	0.906571	1062

4gt12-v1_89

Table 2: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	depth	# gates	# SWAPS	p. success	f	V_Q
no	6	416	658	0	0.768	0.66623522	2496
minextendrc	9	1172	**1360**	**78**	0.562	**0.44841106**	10548
minextend	9	**1008**	1549	99	**0.601**	0.40972458	9072
base	6	1069	1423	85	0.517	0.3581228	6414

4gt4-v0_72

Table 3: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	depth	# gates	# SWAPS	p. success	f	V_Q
no	6	442	746	0	0.786	0.68007548	2652
minextendrc	9	1352	1592	94	0.452	**0.37749204**	12168
minextend	8	**963**	1736	110	0.498	0.34067243	7704
base	6	1056	**1547**	**89**	**0.532**	0.35703954	6336

4mod5-bdd_287

Table 4: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	depth	# gates	# SWAPS	p. success	f	V_Q
no	7	147	203	0	0.916	0.87474237	1029
minextendrc	9	436	500	33	0.753	0.65935538	3924
minextend	9	**332**	500	33	**0.798**	**0.69281491**	2988
base	7	334	**419**	**24**	0.776	0.67942877	2338

4mod5-v0_20

Table 5: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	depth	# gates	# SWAPS	p. success	f	V_Q
no	5	53	61	0	0.985	0.97145968	265
minextendrc	9	139	142	9	0.944	**0.9092329**	1251
minextend	8	**128**	160	11	0.938	0.88981602	1024
base	6	133	**119**	**8**	**0.947**	0.89871898	714

alu_bdd_288

Table 6: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	7	247	0	165	0.94	0.89851036	1155
minextendrc	8	571	36	495	**0.847**	**0.78096707**	3960
minextend	8	616	41	383	0.846	0.73109047	3064
base	7	**472**	**25**	**360**	0.841	0.71637503	2520

Table 7: Results after 1000 iterations, $t_d = 10\mu s$

Mapper	p. success	f
no	0.823	0.7239169
minextendrc	0.645	0.53390012
minextend	0.635	0.44304306
base	0.619	0.42191365

alu_v0_27

Table 8: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	5	107	0	80	0.98	0.96369032	400
minextendrc	9	**278**	**19**	248	**0.959**	**0.92602273**	2232
minextend	10	296	21	**156**	0.944	0.89032214	1560
base	6	**278**	**19**	214	0.915	0.84492332	1284

Table 9: Results after 1000 iterations, $t_d = 10\mu s$

Mapper	p. success	f
no	0.935	0.88798044
minextendrc	0.86	0.78330212
minextend	0.838	0.71318628
base	0.764	0.6236159

decod24_bdd_294

Table 10: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	6	207	0	144	0.938	0.91098461	864
minextendrc	9	441	26	407	**0.888**	**0.7749599**	3663
minextend	7	468	29	328	0.816	0.73708015	2296
base	6	**405**	**22**	**300**	0.781	0.71803687	1800

Table 11: Results after 1000 iterations, $t_d = 10\mu s$

Mapper	p. success	f
no	0.824	0.76021761
minextendrc	0.697	0.48128717
minextend	0.583	0.47226973
base	0.522	0.46210864

decod24_enable_126

Table 12: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	6	978	0	612	0.894	0.74038417	3672
minextendrc	9	2049	119	1788	**0.831**	**0.57285276**	16092
minextend	10	2184	134	**1440**	0.805	0.50947313	14400
base	6	**1959**	**109**	1446	0.74	0.42630108	8676

graycode6_47

Table 13: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	6	21	0	32	0.995	0.99332325	192
minextendrc	7	111	10	111	0.991	0.98223938	777
minextend	10	102	9	61	0.987	0.97012132	610
base	6	84	7	82	0.991	0.98075312	492

ham3_102

Table 14: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	3	61	0	60	0.987	0.98246387	180
minextendrc	4	115	6	127	0.971	0.95999051	508
minextend	4	115	6	121	0.974	0.96288976	484
base	4	106	5	98	0.973	0.95944625	392

hwb4_49

Table 15: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	5	676	0	439	0.932	0.83574294	2195
minextendrc	8	1513	93	1387	0.836	0.64431638	11096
minextend	8	1486	90	961	0.841	0.61627052	7688
base	6	1342	74	1006	0.793	0.55657097	6036

miller_11

Table 16: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	3	147	0	112	0.974	0.95730728	336
minextendrc	4	300	17	307	0.936	0.89826972	1228
minextend	4	291	16	278	0.941	0.90537705	1112
base	4	282	15	231	0.936	0.89555086	924

mini_alu_167

Table 17: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	5	833	0	516	0.891	0.77613311	2580
minextendrc	8	1805	108	1598	0.843	0.6160112	12784
minextend	8	1832	111	992	0.805	0.53931365	7936
base	6	1697	96	1274	0.774	0.51881554	7644

mod10_176

Table 18: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	5	515	0	327	0.9	0.82976826	1635
minextendrc	7	1199	76	1090	**0.758**	**0.62105388**	7630
minextend	10	1127	68	**687**	0.733	0.60641905	6870
base	6	**983**	**52**	734	0.697	0.56115058	4404

Table 19: Results after 1000 iterations, $t_d = 10\mu s$

Mapper	p. success	f
no	0.738	0.59602509
minextendrc	**0.453**	**0.31989048**
minextend	0.443	0.31320313
base	0.372	0.27839542

mod5adder_127

Table 20: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	6	1583	0	944	0.71	0.45135226	5664
minextendrc	9	3320	193	2878	0.491	**0.1922222**	25902
minextend	10	3779	244	2667	0.548	0.18165444	26670
base	6	**3248**	**185**	**2378**	**0.591**	0.18911191	14268

Table 21: Results after 1000 iterations, $t_d = 10\mu s$

Mapper	p. success	f
no	0.528	0.18188697
minextendrc	0.36	**0.1484162**
minextend	0.399	0.14349585
base	**0.465**	0.12694018

mod5d1_63

Table 22: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	5	69	0	59	0.989	0.98368741	295
minextendrc	8	**195**	**14**	209	0.958	0.93474128	1672
minextend	8	**195**	**14**	**136**	**0.969**	**0.93997349**	1088
base	6	**195**	**14**	146	0.95	0.91002595	876

Table 23: Results after 1000 iterations, $t_d = 10\mu s$

Mapper	p. success	f
no	0.97	0.95187372
minextendrc	0.901	**0.84099717**
minextend	**0.914**	0.83627787
base	0.892	0.7849484

mod8_10_177

Table 24: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	6	1270	0	794	0.858	0.70131629	4764
minextendrc	10	**2674**	**156**	2275	**0.52**	**0.39211003**	22750
minextend	10	2827	173	**1761**	0.411	0.29686116	17610
base	6	2773	167	2006	0.335	0.26106507	12036

Table 25: Results after 1000 iterations, $t_d = 10\mu s$

Mapper	p. success	f
no	0.698	0.42021822
minextendrc	0.244	0.19792409
minextend	0.123	0.14638911
base	0.068	0.16412249

one_two_three_v1_99

Table 26: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	5	383	0	256	0.832	0.78653106	1280
minextendrc	7	887	56	839	0.633	0.59855522	5873
minextend	10	869	54	**530**	**0.729**	**0.62135956**	5300
base	6	**833**	**50**	609	0.662	0.57083541	3654

Table 27: Results after 1000 iterations, $t_d = 10\mu s$

Mapper	p. success	f
no	0.602	0.55524768
minextendrc	0.266	**0.38317882**
minextend	**0.355**	0.33820922
base	0.26	0.31493265

one_two_three_v3_101

Table 28: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	5	203	0	143	0.937	0.88807716	715
minextendrc	8	464	29	440	**0.746**	0.620299	3520
minextend	8	509	34	**302**	0.732	0.63161506	2416
base	6	**428**	**25**	323	0.742	**0.62081173**	1938

Table 29: Results after 1000 iterations, $t_d = 10\mu s$

Mapper	p. success	f
no	0.809	0.69629912
minextendrc	0.411	0.31374806
minextend	0.391	**0.31579028**
base	**0.42**	0.31189591

rd32_v0_66

Table 30: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	4	102	0	83	0.983	0.97241164	332
minextendrc	7	**219**	**13**	195	0.947	**0.91458844**	1365
minextend	7	228	14	**142**	**0.958**	0.91079208	994
base	5	**219**	**13**	169	0.955	0.90759692	845

Table 31: Results after 1000 iterations, $t_d = 10\mu s$

Mapper	p. success	f
no	0.95	0.9176419
minextendrc	0.88	**0.79475368**
minextend	**0.902**	0.77708902
base	0.896	0.77242986

sf_274

Table 32: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	6	2227	0	1359	0.484	0.34974095	8154
minextendrc	7	5116	321	4515	0.0	**0.16778098**	31605
minextend	10	5071	316	**3007**	**0.097**	0.14752778	30070
base	6	**4450**	**247**	3289	0.088	0.15461728	19734

Table 33: Results after 1000 iterations, $t_d = 10\mu s$

Mapper	p. success	f
no	0.108	0.16219308
minextendrc	**0.002**	**0.19857107**
minextend	0.0	0.1458942
base	0.0	0.14493197

sf_276

Table 34: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	6	2224	0	1360	0.472	0.30846996	8160
minextendrc	9	4852	292	4103	0.0	**0.16746873**	36927
minextend	10	4807	287	**2747**	**0.092**	0.14342305	27470
base	6	**4447**	**247**	3280	0.089	0.13928494	19680

Table 35: Results after 1000 iterations, $t_d = 10\mu s$

Mapper	p. success	f
no	0.034	0.15718296
minextendrc	0.0	**0.22111901**
minextend	0.0	0.15992956
base	0.0	0.14842314

sym6_145

Table 36: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	7	11185	0	6759	0.506	0.15429107	47313
minextendrc	8	24658	1497	20984	0.513	**0.22079977**	167872
minextend	10	25756	1619	**14156**	**0.546**	0.12489321	141560
base	7	**21679**	**1166**	15613	0.531	0.12176519	109291

Table 37: Results after 1000 iterations, $t_d = 10\mu s$

Mapper	p. success	f
no	0.513	0.1407412
minextendrc	0.518	**0.24438143**
minextend	**0.543**	0.1533595
base	0.53	0.14274046

xor5_254

Table 38: Results after 1000 iterations, $t_d = 30\mu s$

Mapper	# qubits	# gates	# SWAPS	depth	p. success	f	V_Q
no	6	23	0	36	0.995	0.99375935	216
minextendrc	7	68	5	75	0.984	0.9736118	525
minextend	7	68	5	58	0.958	0.94092446	406
base	6	104	9	92	0.942	0.91559086	552

Table 39: Results after 1000 iterations, $t_d = 10\mu s$

Mapper	p. success	f
no	0.984	0.97720823
minextendrc	**0.952**	**0.91998206**
minextend	0.896	0.84674549
base	0.837	0.77312906

Bibliography

[1] X. Fu, L. Riesebos, L. Lao, C. G. Almudever, F. Sebastiano, R. Versluis, E. Charbon, and K. Bertels, *A heterogeneous quantum computer architecture,* Proceedings of the ACM International Conference on Computing Frontiers - CF '16 (2016), 10.1145/2903150.2906827.

[2] X. Fu, M. A. Rol, C. C. Bultink, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels, *An experimental microarchitecture for a superconducting quantum processor,* in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17 (ACM, New York, NY, USA, 2017) pp. 813–825.

[3] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information,* (2009), 10.1017/cbo9780511976667.

[4] T. E. O'Brien, B. Tarasinski, and L. DiCarlo, *Density-matrix simulation of small surface codes under current and projected experimental noise,* npj Quantum Information **3** (2017), 10.1038/s41534-017-0039-x.

[5] R. Versluis, S. Poletto, N. Khammassi, B. Tarasinski, N. Haider, D. Michalak, A. Bruno, K. Bertels, and L. DiCarlo, *Scalable quantum circuit and control for a superconducting surface code,* Physical Review Applied **8** (2017), 10.1103/physrevapplied.8.034021.

[6] P. J. Coles, S. Eidenbenz, S. Pakin, A. Adedoyin, J. Ambrosiano, P. Anisimov, W. Casper, G. Chennupati, C. Coffrin, H. Djidjev, D. Gunter, S. Karra, N. Lemons, S. Lin, A. Lokhov, A. Malyzhenkov, D. Mascarenas, S. Mniszewski, B. Nadiga, D. O'Malley, D. Oyen, L. Prasad, R. Roberts, P. Romero, N. Santhi, N. Sinitsyn, P. Swart, M. Vuffray, J. Wendelberger, B. Yoon, R. Zamora, and W. Zhu, *Quantum Algorithm Implementations for Beginners,* (2018), arXiv:1804.03719v1 [cs.ET] .

[7] M. J. Dousti and M. Pedram, *Minimizing the latency of quantum circuits during mapping to the ion-trap circuit fabric,* 2012 Design, Automation Test in Europe Conference Exhibition (DATE) (2012), 10.1109/date.2012.6176612.

[8] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, and et al., *Superconducting quantum circuits at the surface code threshold for fault tolerance,* Nature **508**, 500–503 (2014).

[9] C. D. Hill, E. Peretz, S. J. Hile, M. G. House, M. Fuechsle, S. Rogge, M. Y. Simmons, and L. C. L. Hollenberg, *A surface code quantum computer in silicon,* Science Advances **1**, e1500707 (2015).

[10] R. Li, L. Petit, D. P. Franke, J. P. Dehollain, J. Helsen, M. Steudtner, N. K. Thomas, Z. R. Yoscovits, K. J. Singh, S. Wehner, and et al., *A crossbar network for silicon quantum dot qubits,* Science Advances **4**, eaar3960 (2018).

[11] IBM, *Ibm q experience backend information,* `https://github.com/QISKit/ibmqx-backend-information` (2018).

[12] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, *Characterizing Quantum Supremacy in Near-Term Devices,* (2016), arXiv:1608.00263v3 [quant-ph] .

[13] E. A. Sete, W. J. Zeng, and C. T. Rigetti, *A functional architecture for scalable quantum computing,* 2016 IEEE International Conference on Rebooting Computing (ICRC) (2016), 10.1109/icrc.2016.7738703.

[14] T. S. Metodi, D. D. Thaker, A. W. Cross, F. T. Chong, and I. L. Chuang, *Scheduling physical operations in a quantum information processor,* Quantum Information and Computation IV (2006), 10.1117/12.666419.

[15] M. Whitney, N. Isailovic, Y. Patel, and J. Kubiatowicz, *Automated generation of layout and control for quantum circuits,* Proceedings of the 4th international conference on Computing frontiers - CF '07 (2007), 10.1145/1242531.1242546.

[16] T. Bahreini and N. Mohammadzadeh, *An minlp model for scheduling and placement of quantum circuits with a heuristic solution approach,* ACM Journal on Emerging Technologies in Computing Systems **12**, 1–20 (2015).

[17] A. Farghadan and N. Mohammadzadeh, *Quantum circuit physical design flow for 2d nearest-neighbor architectures,* International Journal of Circuit Theory and Applications **45**, 989–1000 (2017).

[18] K. E. C. Booth, M. Do, J. C. Beck, E. Rieffel, D. Venturelli, and J. Frank, *Comparing and Integrating Constraint Programming and Temporal Planning for Quantum Circuit Compilation,* (2018), arXiv:1803.06775v1 [quant-ph] .

[19] A. Lye, R. Wille, and R. Drechsler, *Determining the minimal number of swap gates for multi-dimensional nearest neighbor quantum circuits,* The 20th Asia and South Pacific Design Automation Conference (2015), 10.1109/aspdac.2015.7059001.

[20] R. Wille, O. Keszocze, M. Walter, P. Rohrs, A. Chattopadhyay, and R. Drechsler, *Look-ahead schemes for nearest neighbor optimization of 1d and 2d quantum circuits,* 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC) (2016), 10.1109/aspdac.2016.7428026.

[21] A. Zulehner, A. Paler, and R. Wille, *An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures,* (2017), arXiv:1712.04722v2 [quant-ph] .

[22] M. Y. Siraichi, V. F. d. Santos, S. Collange, and F. M. Q. Pereira, *Qubit allocation,* Proceedings of the 2018 International Symposium on Code Generation and Optimization - CGO 2018 (2018), 10.1145/3179541.3168822.

[23] D. C. McKay, T. Alexander, L. Bello, M. J. Biercuk, L. Bishop, J. Chen, J. M. Chow, A. D. Córcoles, D. Egger, S. Filipp, J. Gomez, M. Hush, A. Javadi-Abhari, D. Moreda, P. Nation, B. Paulovicks, E. Winston, C. J. Wood, J. Wootton, and J. M. Gambetta, *Qiskit Backend Specifications for OpenQASM and OpenPulse Experiments,* (2018), arXiv:1809.03452v1 [quant-ph] .

[24] G. W. Dueck, A. Pathak, M. M. Rahman, A. Shukla, and A. Banerjee, *Optimization of circuits for ibm's five-qubit quantum computers,* 2018 21st Euromicro Conference on Digital System Design (DSD) (2018), 10.1109/dsd.2018.00005.

[25] D. Venturelli, M. Do, E. Rieffel, and J. Frank, *Compiling quantum circuits to realistic hardware architectures using temporal planners,* Quantum Science and Technology **3**, 025004 (2018).

[26] M. J. Dousti, A. Shafaei, and M. Pedram, *Squash,* Proceedings of the 24th edition of the great lakes symposium on VLSI - GLSVLSI '14 (2014), 10.1145/2591513.2591523.

[27] J. Heckey, S. Patil, A. JavadiAbhari, A. Holmes, D. Kudrow, K. R. Brown, D. Franklin, F. T. Chong, and M. Martonosi, *Compiler management of communication and parallelism for quantum computation,* ACM SIGPLAN Notices **50**, 445–456 (2015).

[28] Y. Hwang and B.-S. Choi, *Hierarchical System Mapping for Large-Scale Fault-Tolerant Quantum Computing,* (2018), arXiv:1809.07998v1 [quant-ph] .

[29] D. C. Murphy and K. R. Brown, *Controlling error orientation to improve quantum algorithm success rates,* (2018), arXiv:1810.07813v1 [quant-ph] .

[30] L. Lao, B. v. Wee, I. Ashraf, J. v. Someren, N. Khammassi, K. Bertels, and C. G. Almudever, *Mapping of lattice surgery-based quantum circuits on surface code architectures,* Quantum Science and Technology **4**, 015005 (2018).

[31] J. Preskill, *Quantum computing in the nisq era and beyond,* Quantum **2**, 79 (2018).

[32] S. S. Tannu and M. K. Qureshi, *A Case for Variability-Aware Policies for NISQ-Era Quantum Computers,* (2018), arXiv:1805.10224v1 [quant-ph] .

[33] A. Paler, A. Zulehner, and R. Wille, *NISQ circuit compilers: search space structure and heuristics,* (2018), arXiv:1806.07241v1 [quant-ph] .

[34] A. Paler, *On the Influence of Initial Qubit Placement During NISQ Circuit Compilation,* (2018), arXiv:1811.08985v1 [quant-ph] .

[35] R. Jozsa, *Fidelity for mixed quantum states,* Journal of Modern Optics **41**, 2315–2323 (1994).

[36] N. Moll, P. Barkoutsos, L. S. Bishop, J. M. Chow, A. Cross, D. J. Egger, S. Filipp, A. Fuhrer, J. M. Gambetta, M. Ganzhorn, and et al., *Quantum optimization using variational algorithms on near-term quantum devices,* Quantum Science and Technology **3**, 030503 (2018).

[37] A. C. J. M. G. Lev S. Bishop, Sergey Bravyi and J. Smolin, *Quantum volume,* (2017).

[38] R. Wille, D. Gro, L. Teuber, G. W. Dueck, and R. Drechsler, *Revlib: An online resource for reversible functions and reversible circuits,* 38th International Symposium on Multiple Valued Logic (ismvl 2008) (2008), 10.1109/ismvl.2008.43.

[39] A. JavadiAbhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F. T. Chong, and M. Martonosi, *Scaffcc: Scalable compilation and analysis of quantum programs,* Parallel Computing **45**, 2–17 (2015).

[40] C.-C. Lin, A. Chakrabarti, and N. K. Jha, *Qlib,* ACM Journal on Emerging Technologies in Computing Systems **11**, 1–20 (2014).

[41] N. Khammassi, *Openql, high-level quantum programming language,* (2017), unpublished.

[42] N. Khammassi, I. Ashraf, X. Fu, C. G. Almudever, and K. Bertels, *Qx: A high-performance quantum computer simulation platform,* in *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE ’17 (European Design and Automation Association, 3001 Leuven, Belgium, Belgium, 2017) pp. 464–469.

[43] N. Khammassi, G. G. Guerreschi, I. Ashraf, J. W. Hogaboam, C. G. Almudever, and K. Bertels, *cQASM v1.0: Towards a Common Quantum Assembly Language,* (2018), arXiv:1805.09607v1 [quant-ph] .