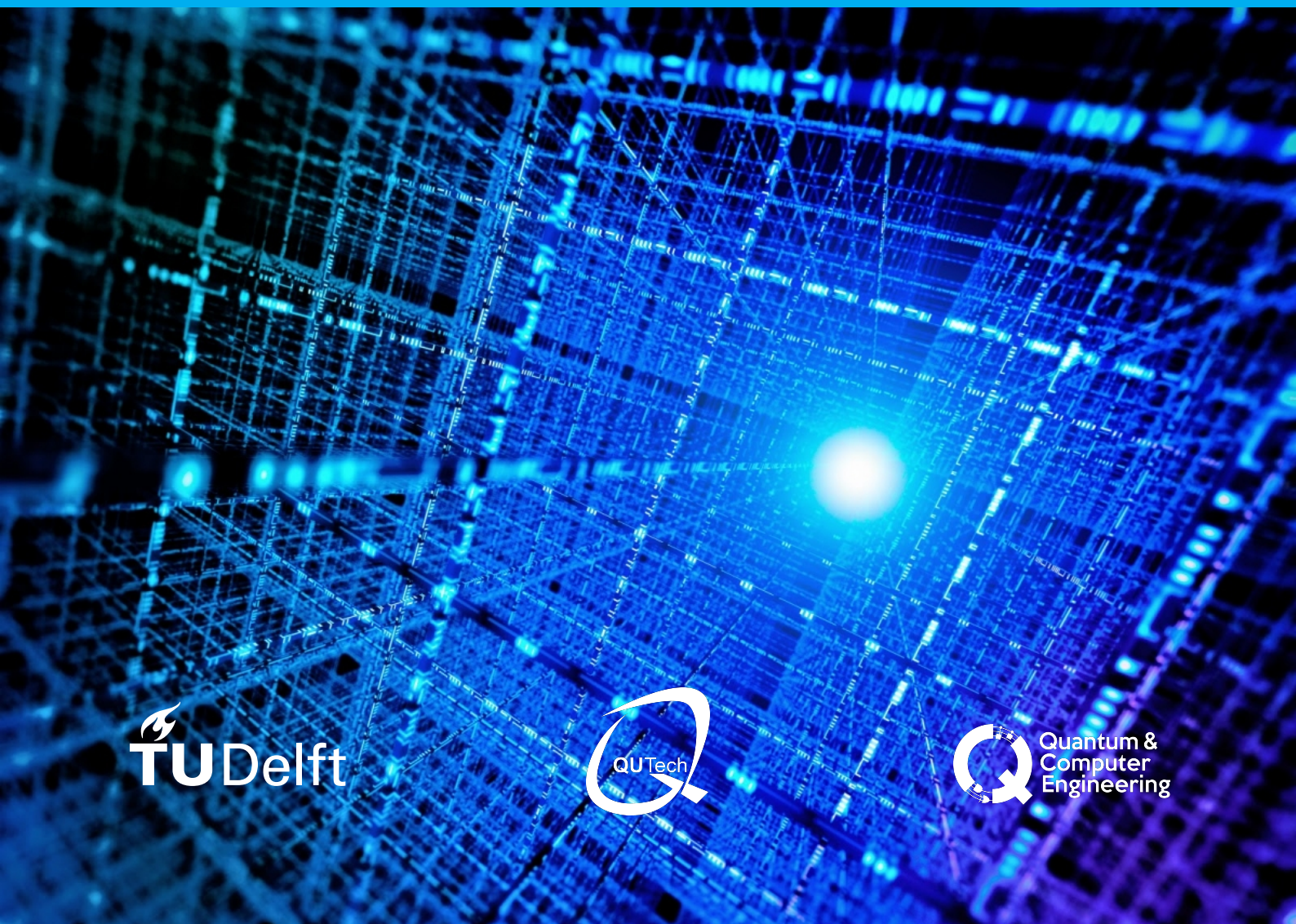
Mapping of quantum algorithms on a quantum chip

2D topology with nearest neighbor interaction

S.G. van Wee



Mapping of quantum algorithms on a quantum chip

2D topology with nearest neighbor interaction

by

S.G. van Wee

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on the 15th of December 2017.

Student number:	4007735	
Project duration:	February 1, 2017 – December 15, 2017	
Thesis committee:	Prof. dr. ir. K.L.M. Bertels	TU Delft
	Dr. ir. C.G. Almudever,	TU Delft supervisor
	Dr. M. Möller	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

QuTech

Quantum & Computer Engineering

Abstract

Quantum algorithms can be described by quantum circuits which consist of quantum bits (qubits) and quantum gates. Such a circuit description assumes that any kind of interaction between qubits is possible. However, quantum chips have limited qubits connectivity only allowing, for instance, nearest-neighbor (NN) interactions. That means, qubits need to be placed in adjacent positions for performing a two-qubit gate. In this thesis, a routing algorithm is proposed where physical qubits or planar-based logical qubits are routed to obey this nearest neighbor constraint in a 2D qubit topology. This algorithm tries to minimize the circuit latency or communication overhead.

The proposed routing algorithm is based on a sliding window principle. Different paths, found by using an adapted breadth-first search algorithm, are evaluated based on the interleaving of the corresponding routing instructions, e.g., SWAP operations with previous instructions, and by looking at the disordering of future qubits. The path that will add the lowest number of cycles to the algorithm is then selected and efficiently inserted with the rest of the instructions. This process continues until all the instructions inside the quantum algorithm obey the nearest neighbor constraints.

The routing algorithm is tested for several real quantum algorithms taken from QLib and ScaffCC, as well as for random generated benchmarks. Taking different alternative paths into account and evaluating those paths for possible interleaving with previous instructions, always has a positive effect to minimize the number of added cycles. The results concerning the evaluation of the disordering of future qubits could have a positive or negative effect on the circuit latency depending on the quantum circuit.

Preface

Two years ago I started my master program in Electrical Engineering with the specialization in Microelectronics. After studying for half a year, I came in direct contact with the field of quantum computing due to the electronics for quantum computation course. At that moment I immediately knew that I wanted to contribute to the field of quantum computing by doing my master thesis in this field.

I would like to express my great appreciation to my thesis supervisor Prof. Koen Bertels and daily supervisor Carmina Almudever. They gave me not only the opportunity to do my master thesis project in the field of quantum computing but also the possibility to involve in the overall contribution of the entire Quantum and Computer Engineering (QCE) group. The multiple discussions on all the related topics were always interesting and motivating to deliver a nice contribution to this research field; I truly felt part of the team.

I would like to thank everybody in the QCE group for all the discussions and fun moments in and around the 10^{th} floor of the faculty of EEMCS. Special thanks to Koen and Carmina for the great support and the in-depth discussions and feedback to help me guide to get the maximum out of my research project. Also many thanks for the great flexibility in scheduling the work that allowed me to participate in two solar car events with the Nuon Solar Team the past year. Also, I would like to separately thank Leon Riesebos and Hans van Someren for all the spontaneous help and discussions that paved the way to speed-up the research. You always had time to answer my questions. Leon and Andreas Papageorgiou also thank you for all the fun moments and the various exciting side projects. Last but not least, Lingling Lao, thank you for fully test and review the routing algorithm, so it was even able to give a contribution to your currently unpublished paper.

Finally, I would like to show my gratitude to my girlfriend, my parents and my brother and sisters and friends that have been standing by my side for the last years. I have always been able to count on them during the good and the bad times.

Thank you all very much!

S.G. van Wee
Delft, December 2017

Contents

List of Figures

List of Tables

List of Acronyms

ALAP	As late as possible
ASAP	As soon as possible
BFS	Breadth-First Search
BVSearch	QLib: Bernstein-Vazirani Search
CNOT	Controlled NOT-gate
CZ	Controlled Z-gate
DFS	Depth-First Search
EPR	Einstein–Podolsky–Rosen
ESM	Error Syndrome Measurement
NN	Nearest Neighbor
FT	Fault-Tolerant
LD	Logic Depth, total sequential number of cycles
LLD	Lower Logical Depth
NoC	Network on Chip
QASM	Quantum Assembly Language
QEC	Quantum Error Correction
QECC	Quantum Error Correction Cycle
QME	QLib: Quantum Modular Exponentiation
SC	Surface Code
SC-17	Surface Code with 17 qubits, so-called Ninja-star
SCC	Surface Code Cycle
VBE adder	QLib: Vedral-Barenco-Ekert carry-ripple adder

1 Introduction

In this chapter, the need to map quantum circuits on a qubit chip is explained. The current background in quantum computing is introduced in Section 1.1 together with a problem definition and the document organization in Section 1.2 and 1.3, respectively.

1.1. Background

In a world where the need for computational power grows every day, the demand for a new, faster technology has arisen. Where the current CMOS technology is pushed against its limits, this technology is getting close to fundamental physical problems. With decreasing transistor sizes, certain quantum phenomena such as tunneling, start to play an important role and limit certain developments. Therefore attempts are being made to use the quantum phenomena such as superposition and entanglement to develop a new type of computer, the quantum computer. When using these quantum phenomena in a controlled environment, the computational power could grow exponentially with the number of data elements (qubits).

To fully utilize all this computational power new quantum algorithms need to be developed. Before a quantum algorithm can be executed on the quantum chip, different steps need to be performed. These steps are illustrated as layers in the system stack proposed by QuTech at Delft University of Technology [2] as shown in Figure 1.1. The top layer of this stack is the quantum algorithm, that is described in a high-level quantum programming language such as Quipper [9] and OpenQL [13]. This quantum algorithm can be represented by a quantum circuit. At the compiler, the quantum circuit is decomposed, optimized and mapped on the underlying quantum chip. The output of the compiler is a series of Quantum Assembly Language (QASM) based instructions. This QASM is a technology independent language that describes the quantum circuit and can be simulated on a quantum simulator or executed on the quantum chip. The QASM instructions, which belong to the Quantum Instruction Set Architecture (QISA), are sent to the microarchitecture and translated into the corresponding hardware dependent pulses/signals. These pulses/signals are sent through the quantum to classical interface to the quantum chip [7].

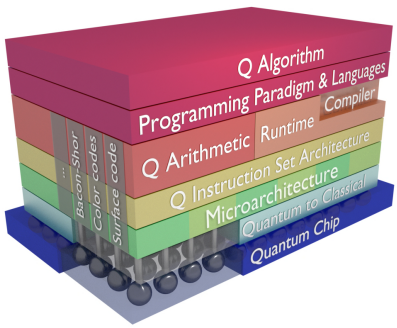


Figure 1.1: The system stack design of the quantum computer architecture [4].

1.2. Problem definition

As previously mentioned, quantum algorithms can be described by quantum circuits. The quantum circuits consist of qubits and quantum gates operating on those qubits. This description of quantum circuits is quantum hardware agnostic and usually assumes that all qubits can perform any operation with any other qubit (full qubit connectivity). However, at the lowest layer of the stack, the qubit chip, some hardware constraints should be taken into account when running a quantum algorithm.

Most quantum technologies propose to arrange the qubits in a 2D grid with only nearest neighbour interaction [31]. Due to the NN hardware constraint, only qubits which are adjacent can interact with each other [24], perform a so-called two-qubit gate. This NN constraint makes the actual interaction inside the 2D-qubit plane a major challenge for quantum computers as qubits need to be moved next to each other to perform a two-qubit gate. This extra movement will add communication overhead to the quantum circuit. This communication overhead can be expressed as the added number of movement operations and the increased circuit latency. Depending on the quantum algorithm, some or a lot of the execution time in a large-scale quantum computer is used for this communication or movement of quantum states [2] [4] [8]. An optimal solution is then needed which minimizes the communication overhead while obeying the NN constraints of the quantum chip.

In this thesis, a routing algorithm is introduced to map quantum circuits on a qubit plane to comply with the NN interaction constraint. The routing algorithm will find the paths for moving qubits around and inserts the required (routing) operations. As the qubits will decohere over time, the proposed routing algorithm will try to minimize the total execution time of a quantum circuit. The routing algorithm is based on a sliding window principle in which the two-qubit gates are evaluated to see if a transportation path needs to be inserted, and what the shortest path is, based on the added number of cycles/time concerning the original circuit. In this thesis, the routing is performed just after the compiler decomposes, optimizes and schedules the quantum circuit.

In short, the goal of this thesis is to develop a routing algorithm, which inserts extra qubit movement operations, to respect the NN constraint of the 2D-qubit lattice, while minimizing the communication overhead with respect to time. In this thesis the following research questions will be answered:

- What are the differences between routing classical data and routing quantum states?
- What are the possibilities for routing quantum states?
- Which methods can be used for transporting qubits?
- What are good metrics for classifying quantum circuits?
- What is the difference between routing physical and planar-based logical qubits?
- What are good techniques and the limitations when routing quantum states?

A conclusion will be formed based on the results of these research questions.

1.3. Organization

The thesis is structured according to the research questions stated above.

In chapter 2, a background in quantum computing is presented. In this chapter the basics of quantum computing are introduced. Quantum bits or qubits, and single-qubit as well as multi-qubit gates are discussed. Furthermore, the principle of fragile qubits is introduced and why this leads to the need of quantum error correction.

In chapter 3, first the different steps required for mapping a quantum circuit are explained. From this mapping the routing is investigated. Therefore the difference between routing data on a classical computer and on a quantum computer is analyzed. In this chapter routing methods used in classical computers which could be useful in quantum computers are explained. Different transportation methods in a qubit plane are also reviewed.

In chapter 4, the proposed algorithm for routing quantum states is presented. First, the QASM language used to describe a quantum circuit, and how the input file to the routing algorithm is generated by the compiler, are introduced. Then, the different functions of the routing algorithm are discussed, as well as the rescheduling and verification steps. Also is shown how the routing algorithm can be adjusted from physical to logical planar-based qubit routing. Finally, the chapter concludes with a description of all the metrics used to evaluate the impact of routing on different quantum algorithms.

In chapter 5, several quantum algorithms (benchmarks) are introduced and characterized. These benchmarks are routed using the routing algorithm proposed in this thesis. Random generated quantum benchmarks, as well as some real quantum algorithms taken from QLib and ScaffCC, are used to assess the performance of the routing algorithm. Next, some performance measurements for routing logical planar-based qubit circuits. This chapter concludes with a small comparison with other routing algorithms found in related literature.

In chapter 6, the scalability and implementabilty of the routing algorithm are discussed as well as possible improvements.

In chapter 7, the conclusions are drawn, and recommendations for future work are given.

2

Background

In this chapter, the basic background knowledge of quantum computing is explained. In Section 2.1 what a qubit is and how qubits (or quantum states) can be altered by applying quantum gates is explained. In Section 2.2 the principles of Quantum Error Correction and Fault-Tolerance to achieve reliable quantum computation are introduced.

2.1. Quantum computing

The fundamental elements of a quantum computer are quantum bits, also known as qubits. These qubits can be represented in a physical system, just as classical bits. However, qubits have two key features that classical bits don't have, superposition and entanglement. This superposition and entanglement are explained in Subsection 2.1.1. How the state of a qubit can be changed by using quantum gates is explained in Subsection 2.1.2.

2.1.1. Quantum bits

A bit can only take one binary value of 0 or 1 at any point in time. In a quantum computer, these binary values, 0 and 1, are called states. These two states are defined as: $|0\rangle$ and $|1\rangle$, a so-called Dirac bra-ket notation. A qubit can be in state $|0\rangle$ or state $|1\rangle$ but also in a superposition of both states. A qubit state is defined as a linear combination of these two states and is represented by $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, in which $\alpha, \beta \in \mathbb{C}$ (are complex numbers) and are called probability amplitudes. The sum of these probability amplitudes should always be 1, which means $|\alpha|^2 + |\beta|^2 = 1$.

Due to the probabilistic nature of qubits, a qubit will give a probabilistic result when measured in the computational basis. This probabilistic result means that there is a probability of $|\alpha|^2$ to measure a $|0\rangle$ state and a $|\beta|^2$ to measure a $|1\rangle$ state (when measured in the Z-basis). When a measurement takes place, the qubit collapses to one of its basis states and loses the probabilistic information.

Qubit its superposition can also be represented as a matrix. In this notation, the states are then represented as a unit column vector. The so-called Hilbert space, $\mathscr{H}$, is made from these vectors as an orthonormal basis, also known as the computational basis. This vector notation is shown in equation 2.1. This matrix representation will be used in Subsection 2.1.2 to explain how these states can be changed by performing quantum gates.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \alpha\begin{bmatrix}1\\0\end{bmatrix} + \beta\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}\alpha\\\beta\end{bmatrix} \tag{2.1}$$

The orthonormal basis can also be visualized. The orthonormal vectors span a surface in the shape of a sphere, the so-called Bloch sphere. Two vectors can be written as shown in equation 2.2. With this notation the qubit state is represented as a single point on a sphere as can be seen in Figure 2.1

$$|\psi\rangle = cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}sin\left(\frac{\theta}{2}\right)|1\rangle \tag{2.2}$$

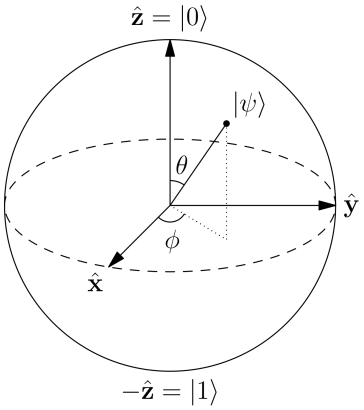


Figure 2.1: The Bloch sphere.

When multiple qubits are used, the global state can be represented as a column vector which satisfies $2^n = N$, in which n is the number of qubits and N the number of dimensions in the Hilbert space. Due to the superposition principle of the qubit, the Hilbert space grows exponentially with the number of qubits.

The other feature of qubits is that qubits can entangle with each other. When this entanglement takes place, the total system cannot be described as a product of the individual qubits spaces. An example of a non-entangled state is shown in equation 2.3, an entangled state is shown in equation 2.4.

$$\left|\psi\right\rangle = \frac{1}{\sqrt{2}}|11\rangle + \frac{1}{\sqrt{2}}|01\rangle = |1\rangle \otimes \frac{1}{\sqrt{(2)}}(|0\rangle + |1\rangle) \tag{2.3}$$

$$\left|\psi\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{2.4}$$

2.1.2. Quantum gates

The state of a qubit can be changed by applying quantum gates. These quantum gates can be represented as $2^n * 2^n$ matrix, where n is the # of qubits the quantum gate acts on. When a gate is applied, the operation is a simple matrix-vector multiplication. Because a qubit state vector is a unit vector ($|\alpha|^2 + |\beta|^2 = 1$), the matrices representing the quantum gates must be unitary. This unitary means that equation 2.5 must be satisfied for all quantum gates. Due to this property, all quantum gates are reversible.

$$U * U^\dagger = U^\dagger * U = I \tag{2.5}$$

Single-qubit gates

A single quantum gate acts on a single qubit. This gate is represented as a 2 x 2 unitary matrix. An example of a quantum gate (the Pauli-X gate) is shown in equation 2.6. The quantum circuit of a Pauli-X gate is shown in Figure 2.2. This gate will perform a bit-flip, transform a quantum state $\alpha|0\rangle + \beta|1\rangle$ to $\beta|0\rangle + \alpha|1\rangle$. The most commonly used single-qubit gates and their corresponding matrix representation can be found in Table 2.1.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.6}$$

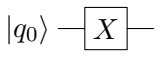


Figure 2.2: Quantum circuit of a Pauli-X gate.

Table 2.1: Quantum gate circuit and matrix representations [2].

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

Multi-qubit gates

Multi-qubit gates can be classified into two groups, a two-qubit gate or a gate which acts on more than two qubits. The most used two-qubit gate is the CNOT gate also called is a controlled-NOT gate. The matrix representation of the CNOT gate and its quantum circuit are shown in equation 2.7 and Figure 2.3, respectively, where $|q_0\rangle$ is the control qubit and $|q_1\rangle$ is the target qubit. A CNOT gate works as follows, when the control qubit is in the $|1\rangle$ state, on the target the Pauli-X gate will be performed; that is, the target qubit will be flipped when the control qubit is in state $|1\rangle$.

$$CNOT = \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{bmatrix} \tag{2.7}$$

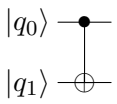


Figure 2.3: Quantum circuit of a CNOT gate.

When combining three CNOT gates, as shown in Figure 2.4, a SWAP gate is performed. The matrix representation is shown in equation 2.8. A SWAP gate exchanges the state of two qubits as shown in Figure 2.4. SWAP gates can then be used for transporting quantum information (moving qubits around). In this thesis, the SWAP gates will be used as a movement operation, this will be discussed in chapter 3.

$$SWAP = \begin{bmatrix}1 & 0 & 0 & 0\\0 & 0 & 1 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\end{bmatrix} \tag{2.8}$$

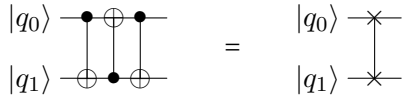


Figure 2.4: Quantum circuit of a SWAP gate.

Other examples of two-qubit gates can be found in, for example, Michael A. Nielsen and Isaac L. Chuang [24].

Universal set of gates

Universal quantum gates is any set of gates to which any operation possible on a quantum computer can be reduced, that is, any other unitary operation can be expressed as a finite sequence of gates from the set. In this thesis, the assumption is made that any quantum circuit is decomposed into the universal set of gates listed in Table 2.1, the CNOT gate and the SWAP gate.

2.2. Reliable quantum computation

Qubits are fragile elements concerning its quantum state. Due to for example noise and interaction with the environment, the state of a qubit may change and loose its information. The probability that the qubit is in its correct state, the so-called decoherence of a qubit, will decay over time. This decoherence means that as a quantum circuit runs for a longer period, the probability of the quantum circuit giving a correct result decreases. Besides the decoherence of qubits, quantum gates are not perfect and also introduce errors. Therefore, quantum systems need to be protected against errors to be able to perform reliable quantum computations. To make quantum computing fault-tolerant (FT), Quantum Error Correction (QEC) and FT mechanisms are used.

2.2.1. Quantum Error Correction (QEC)

QEC is inspired by classical error correction. In classical error correction, information is protected against possible errors by; 1) encoding data in which (odd) copies of the same bit are made, 2) detecting errors by measuring out the encoded data (majority voting), and 3) and correcting the original data [24]. The same steps can be used for protecting qubits when using QEC. In quantum error correction, quantum data is encoded in logical qubits. A logical qubit is a more reliable unit composed by several unreliable physical qubits. Note that this encoding step is performed by entanglement because quantum bits cannot be copied (non-cloning theorem). To detect if an error has occurred, extra qubits are needed; a qubit state cannot be measured directly as the superposition will fall into one of its basis states, destroying its information. These extra qubits, so-called ancilla qubits, perform parity checks between the data qubits of the logical qubit. Ancilla qubits are measured giving a $+1$ or -1 measurement results, also called error syndromes. The error syndromes are used to detect potential errors. When an error is detected, the type of error and on which qubit, the logical qubit can be corrected to restore its original state.

One of the most promising QEC codes is the Surface Code (SC), due to its simple structure, the 2D array with only NN-interaction, and the high error threshold [6]. In SC, qubits are arranged in a 2D lattice as shown in Figure 2.5. In this figure, the red and green dots represent the ancilla qubits, and the white dots are the data qubits. The data qubits store the information, whereas the 'X' (in red) and 'Z' -ancilla qubits (in green) are used for detecting phase-flip (X) and bit-flip (Z) errors, respectively. As shown in Figure 2.5, each ancilla qubit connects and can interact with its four neighboring data qubits. The ancilla qubits detect possible errors by performing error syndrome measurements (ESM). The corresponding error syndrome measurement circuits are shown in Figures 2.6 and 2.7 respectively, in where ancilla qubits entangle with its neighboring qubits and are measured in the end. The measurements outcomes, or error syndromes ($+1$ or -1), indicate possible errors inside the lattice (on one or more ancilla and/or data qubits). To detect or identify what errors have occurred, decoding algorithms are used. These decoding algorithms use several rounds of ESMs to filter out for example measurement errors, to be able to identify any error in any qubit. Therefore, QEC is a resource expensive and complex problem [6] [30].

As previously mentioned, the qubit information/state is encoded in a logical qubit. A logical qubit consists of multiple non-perfect physical qubits. The number of physical qubits per logical qubit depends on the code distance. The code distance is defined as the minimum number of physical operations required to perform a logical operation [6]. When increasing the code distance, more errors can be detected and corrected, increasing then the reliability of the computation. What the number of detectable errors or distance should be, depends on the required logical qubit error rate and the current physical qubit error rate (technology error rate). In Figure 2.5 a logical qubit consisting of 17 physical qubits is shown (in purple). This logical qubit is a planar-based surface-17 code (SC-17) with distance 3, meaning that a maximum of 1 bit-flip and 1 phase-flip errors can be detected and corrected per error correction cycle. How multiple ESMs are used to detect and correct errors in SC-17 can be found in other literature [6] [11] [30].

Another method of building logical qubits in surface code is using a defect based approach [6]. In defect based SC, logical qubits are encoded by introducing 'holes' or 'defects' in the lattice. To create such a hole, some ancilla qubits stop performing the ESMs. In this thesis on planar-based logical qubits will be discussed.

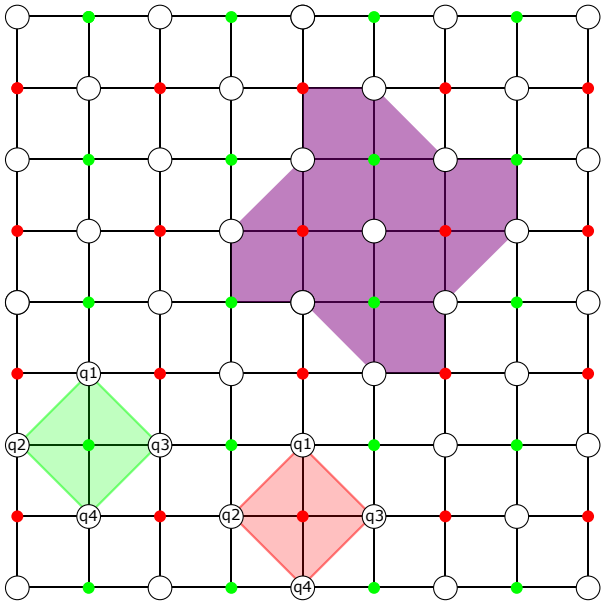


Figure 2.5: 2D qubit lattice with planar-based surface coding. The purple part is a 17-qubit surface code. The green dots are the X-ancilla qubits, the red dots the Z-ancilla qubits and the white dots are the data qubits.

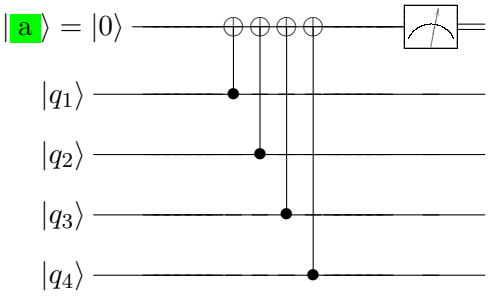


Figure 2.6: Error syndrome measurement circuits for X ancillas.

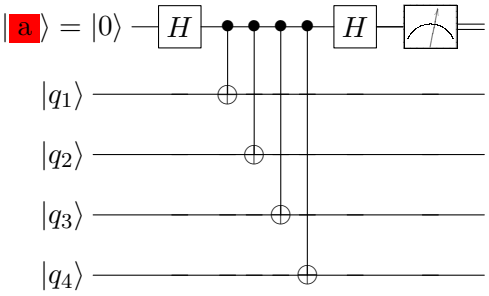


Figure 2.7: Error syndrome measurement circuits for Z ancillas.

2.2.2. Fault-Tolerant (FT) quantum computation

Quantum error correction (encoding and detection) is not sufficient for having reliable quantum computation. In addition, logical quantum gates (gates on logical qubits) need to be performed in a fault-tolerant (FT) way. A procedure is defined to be FT if an operation is performed on the logical qubit state without losing the encoding, which protects the logical qubit from errors, and how to perform error correction when the gates used on the logical qubit are noisy [24]. A protocol which performs these tasks is called FT. The way to perform a quantum gate on planar-based logical qubits differs from the way of applying a gate on physical qubits due to this FT requirement.

Single-qubit gates

In planar-based surface code, some single-qubit gates such as Pauli-gates or Hadamard can be applied transversally, i.e., applying bitwise physical operations on a subset of the data qubits, and then performing d-rounds of ESM to detect errors. For instance, performing a Pauli-X gate on an SC-17 logical qubit, three physical X gates on data qubits need to be applied as shown in Figure 2.8. This is step is illustrated by the X inside the data qubits. After d-rounds of ESMs, the logical qubit is then brought to its new state. The same procedure holds for the Pauli-Z gate. Only the physical Pauli-Z gates are performed on other data qubits as illustrated in Figure 2.10.

Other single-qubit logical gates- e.g., Hadamard gate or T and S gates are more complex to implement and require the use of other techniques or procedures such as lattice surgery [11] or state distillation [6].

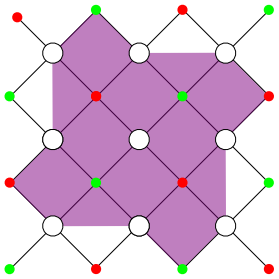


Figure 2.8: Original planar-based SC-17 logical qubit.

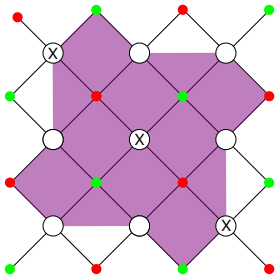


Figure 2.9: Logical Pauli-X gata on a SC-17 logical qubit.

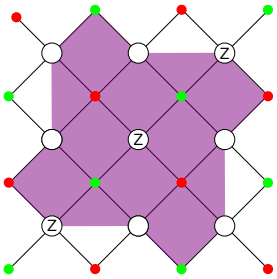


Figure 2.10: Logical Pauli-Z gata on a SC-17 logical qubit.

Multi-qubit gates

To perform a CNOT between two logical qubits in a 2D array with NN constraint, a procedure called lattice surgery is needed [11]. This logical CNOT gate is realized by merging and splitting lattices and requires an extra ancilla logical qubit 'A' as shown in Figure 2.11. It works as follows; 1) lattices 'A' and 'C' are merged and then split; 2) lattices 'A' and 'T' are merged and then split; and 3) measure 'A'. The merging and splitting operations are implemented by performing ESM on the 'merged' lattices and the 'separated lattices', respectively. Note that the control (C) and target (T) qubits need to be aligned at a 90-degree angle with an ancilla logical qubit in between as can be seen in Figure 2.11. Therefore, this may require that if a logical qubit that needs to interact and is not placed in such positions (close to each other), need to be moved. The procedure is applicable for general rectangular SC lattice shapes, as can be seen in Figure 2.11, as well for complex SC structures such as SC-17, as can be seen in Figure 2.12.

As we will see in next chapters, the routing algorithm proposed in this thesis can be used for routing physical qubits as well as planar-based SC logical qubits.

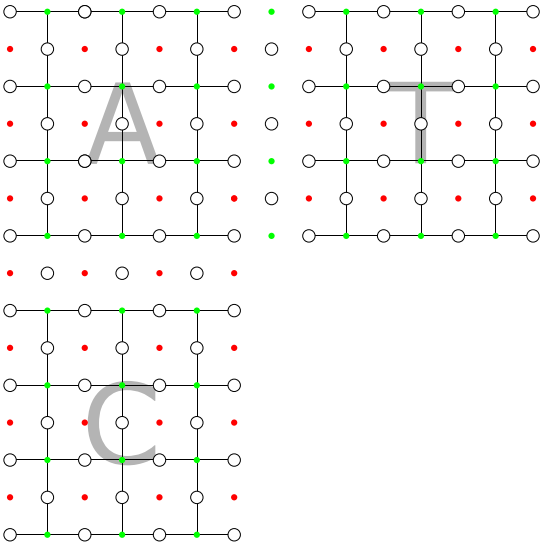


Figure 2.11: CNOT operation of planar-based qubit through lattice surgery. The control, 'C', and target, 'T', interact with each other by merging and splitting with the intermediate surface, 'A'.

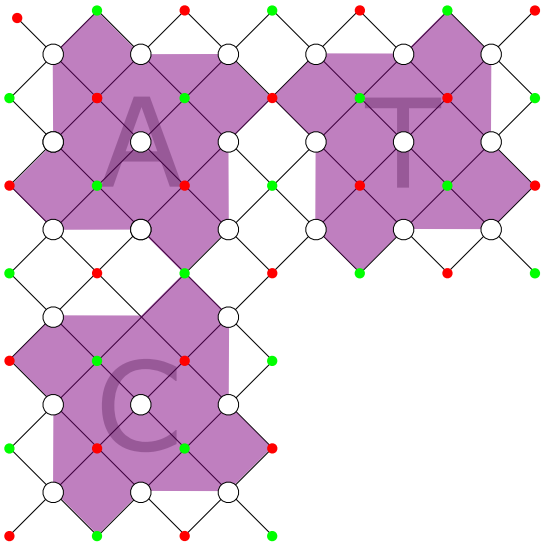


Figure 2.12: CNOT operation of a planar-based SC-17 logical qubit qubit through lattice surgery. The control, 'C', and target, 'T', interact with each other by merging and splitting with the intermediate qubit, 'A'.

3

Mapping of quantum circuits

This chapter starts by explaining in Section 3.1 the different steps required for mapping a quantum algorithm, which includes: scheduling of operations, placement of qubits, and routing of qubits. As mentioned previously, only the routing of qubits will be investigated in this thesis. Various routing techniques in a classical and quantum computer are discussed in sections 3.2 and 3.3 respectively. Following, the transportation of physical as well as logical qubits is covered in Section 3.4. This section concludes with a selection of transportation methods used in the proposed routing algorithm in chapter 4.

3.1. The mapping process

When mapping a quantum algorithm on a qubit plane, some constraints need to be obeyed which were not there when describing the quantum algorithm. As mentioned in the introduction, a quantum algorithm can be described by a quantum circuit when the circuit model is adopted as a computational model. Such a quantum circuit description does not consider the physical hardware connectivity between qubits and does not have any notion of gate execution time. To be able to execute such a quantum circuit on a quantum chip, mapping is required. This mapping consists of: scheduling of operations, placement of qubits and routing of quantum states. [2].

Scheduling of operations

The scheduler will try to optimize the latency (total execution time) of the quantum circuit by maximizing the parallelism of instructions. The scheduler does this optimization by for example generating a dependency graph, deriving the critical path and scheduling operations, e.g., as late as possible (ALAP). Assumed that optimizations like gate decomposition, which splits complex gates into simpler gates, and removing sequences of gates that together are an identity operation, have been applied before scheduling. These optimizations may lead to a higher parallelism and a shorter latency of the quantum circuit.

Placement of qubits

In addition, 'virtual' qubits, the qubits specified in the quantum circuit, need to be placed on a specific position on the quantum chip. Quantum chips have a limited connectivity between the qubits, and only adjacent qubits can interact. When qubits are not next to each other, extra transportation is needed. Therefore an initial placement may lead to less transportation. This initial placement could, for example, be designed to minimize the Manhattan distance between the qubits. The result of this initial placement is an initial mapping of the qubits of the quantum circuit on the physical qubits.

Routing of qubits

After an initial placement, it is most likely that not all qubits that need to interact are adjacent. Therefore a routing algorithm is required to move qubits around and place them in NN positions when required. The routing algorithm will find a communication path and will insert the required movement operations.

The rest of this chapter will focus on the routing of qubits.

3.2. Routing of classical data

The problem of routing quantum states or qubits is very similar to the routing in the classical Network on Chip (NOC) approach. In NOCs, several processing units or nodes are connected in a specific configuration, e.g., star, mesh, ring, etc. and data (or packets) need be sent from different sources to different destinations. This data movement requires a routing algorithm or protocol which defines the path taken by the data in a network with specific characteristics, such as network topology, channel/link bandwidth, and buffer capacity. This NOC approach has two parts related to this thesis: the network topology and characteristics, and the routing algorithm.

3.2.1. Network topology and characteristics

For this thesis, only data transportation on the lowest level, the qubit chip, is explored. As will be explained in Section 3.3.1, the qubits in a qubit chip are arranged in a 2D lattice. The transporting data on a qubit chip is the state of the qubits. The transportation is performed by performing the SWAP operation introduced in Section 2.1.2. A SWAP operation exchanges the state of two adjacent qubits and can be seen as a 'bi-directional' operation.

To compare quantum data transportation with a classical data transportation, the NOC representation can be evaluated. As mentioned earlier, in NOCs the network topology, channel/link bandwidth, and buffer capacity are taken into account. When relating these classical characteristics with a quantum computer, the data transportation in a qubit plane, i.e., SWAP, go in both directions at the same time and is performed in one instruction (bi-directional communication) and has a fixed latency. The topology of the qubit chip can be related to a classical mesh network where each node has four neighbors to form the 2D array.

In a classical computer, buffers are used to store temporary data. By temporarily storing data in buffers, communication channels can, for example, be used by other data, or to maximize the throughput. Comparing the quantum chip to a classical approach, the nodes contain a data element. Only this data element can be moved to the adjacent nodes. For the proposed routing algorithm, there are no buffers used to temporarily store the original data to 'empty' a node.

The major difference between data transportation in a classical versus a quantum computer is that data needs to be brought to a computational unit in a classical computer. While on the qubits, a computational unit/gate can be performed directly. Therefore only the quantum operations that need movement operation are taken into account when evaluating the different classical routing algorithms.

When going to large-scale quantum circuits with multiple qubits, more complex structures may be required. The NOC representation could be more useful when going to these complex structures when for example, separate communication channels and buffers are specified. Inside this thesis, only small and less complex structures are evaluated where no separate communication channels are needed.

3.2.2. Routing algorithm: Finding paths

To find the optimal path to transport data from point A to B different search algorithms have been developed. One of the best-known search algorithms is the Dijkstra algorithm [25] [17]. This search algorithm can find the shortest paths between two nodes in a graph. To find the shortest path a number of steps are specified in the search algorithm. In short, the Dijkstra algorithm will find the shortest paths by first setting the distance at every node (except for the starting node) to infinity. The distance of the starting node will be set to zero and selected as the current node. For this current node, its neighboring nodes are considered. For each neighboring node, the distance from the start is stored. The start node is marked as visited and will never be checked. For the next step, the node with the lowest distance is selected as the new current node. For the current node, the neighboring nodes are considered. Again the distance from the start node is stored in the nodes, and the current node is marked as visited. If a shorter path is found, the lowest distance from the node to the start node is stored. This process is continued until the target is reached.

When introducing this algorithm for a quantum chip, the weights of the edges specified in the Dijkstra algorithm will be constant. Therefore less complex routing algorithms can be used based on Network on Chip (NOC) routing [26]. These routing algorithms can be divided into oblivious routing algorithms and adaptive routing algorithms. Oblivious routing algorithms have no information about the condition of the links (in quantum computing described as the connection between qubits) and nodes (in quantum computing described as the qubits), i.e., they don't know whether nodes are occupied or whether there is other traffic inside the network. Adaptive routing algorithms do have information about the network. Depending on the exact routing algorithm, it will take into account congestions of nodes and other traffic inside the network. Some examples of both routing algorithms with their main advantages and disadvantages are given in the next subsections.

Oblivious routing algorithms

Oblivious routing algorithms can be classified into different groups according to V. Rentala [26]: dimension order routing, turn models, deterministic routing and stochastic routing. An example of a dimension order routing algorithm is XY-routing. XY-routing will try to find a path by first going in the X(Y) direction and then in the Y(X) direction until it reaches its target. The turn model algorithm uses the dimension order routing but allows extra turns which may lead to a longer path. Deterministic routing algorithms route data along a fixed path. Stochastic routing algorithms will send its data with a certain "time to live". This time to live will specify how long a data element is allowed to move around in the network. The data will be removed from the network after this time is over.

Adaptive routing algorithms

Adaptive routing algorithms can also be classified into different groups according to V. Rentala [26]: minimal adaptive routing, fully adaptive routing and turnaround routing. Minimal adaptive routing uses the shortest path. The shortest path is not only selected based on distance but also based on congestions. Fully adaptive routing algorithms select a path without any congestions. This path may not be the shortest path. Turnaround routing allows multiple senders and receivers in a network and also bi-directional communication. This communication means that data may also return to its origin.

These classical routing algorithms all try to find the shortest path based on basic single path strategies. Each routing step evaluates one or multiple possible edges inside the connectivity graph. The adaptive routing algorithms try to use these basic strategies and uses the knowledge of the state of edges and nodes to avoid congestions. Some even try to optimize bi-directional communication which might be useful when designing a quantum routing algorithm due to the SWAP operation.

The main disadvantages of these routing strategies is that it will only evaluate one or several options depending on the routing algorithm. Therefore a higher level pathfinding approach might be useful to find more and therefore evaluate more alternative paths.

3.2.3. Possible data loss with routing

In classical routing, there is the possibility that the data which is sent will never reach its destination. Deadlock, livelock, and starvation may be a cause for data not reaching its destination.

A deadlock occurs if, for example, two nodes need each others resources at the same time. For the two nodes, the path they want to take contains both nodes, causing them to wait for each other to be released. See Figure 3.1 for an illustration of a deadlock problem.

A livelock occurs if, for example, data is routed in a ring. The data is constantly sent to a neighboring node with the shortest path. Due to the blocking surrounding nodes, the data will stay in this area. See Figure 3.2 as an illustration of a livelock problem.

Starvation occurs when data never reaches its destination. This starvation may, for example, occur if a stochastic routing algorithm is used. The data will be removed from the network after a certain time-to-live has passed. This removal could, for example, happen if the data needs to travel further than originally intended.

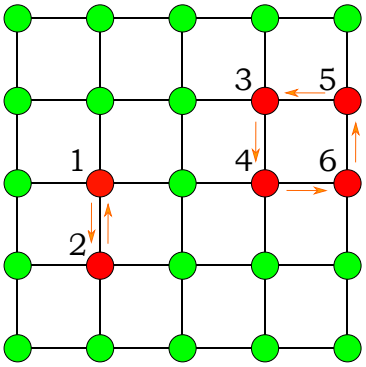


Figure 3.1: Two illustrations of a deadlock. Data point 1 and 2 both want to use each others resources for transportation. The same hold for the ring which consists out data points 3 through 6.

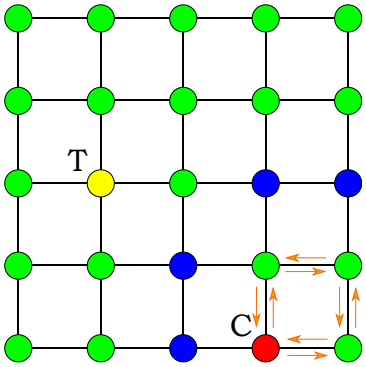


Figure 3.2: Illustration of a livelock. The data point source (C) is not able to find a path to reach the target (T) due to the occupied data points in blue.

3.2.4. Graph theory

The NOC routing algorithms evaluate single or multiple well-chosen alternatives. Nevertheless, there is the possibility that data loss occurs during routing. Therefore a higher level pathfinding approach can potentially be used to find all possible shortest path. The pathfinding approach can be found among the classical graph search algorithms [16]. These search algorithms are specified for searching inside a tree or graph. The search algorithms will start from a single node and will continue along the different branches until they find the target. The search algorithms can possibly be extended to find all shortest path inside a tree or graph.

An example of a search algorithm is Depth-First Search (DFS). DFS will search for the destination by first exploring a single branch up to its maximum depth before going to a side or new branch. In Figure 3.3 the various steps of a DFS are illustrated. Just as in the classical routing algorithms, the underlying connectivity is a grid. Therefore the DFS algorithm will try to find the target by evaluating the various nodes. All edges have the same weight, and if a node is visited twice, this branch will then end. The DFS algorithm will immediately stop when the target has been found. In this case, the search will continue till it has found node 14.

Another example of a search algorithm is Breadth-First Search (BFS). BFS will work in the same way as DFS except that first all branches are evaluated up to the same depth. First, the neighbor nodes are explored before going to a lower depth of neighbors. In Figure 3.4 the various steps of a BFS are illustrated. The BFS algorithm will immediately stop when the target (again node 14) has been found.

In this thesis, the focus will be on the routing of quantum states in 2D lattices with only a low number of qubits, up to 49 qubits. Therefore all possible paths can and will be evaluated, by using the BFS algorithm just described.

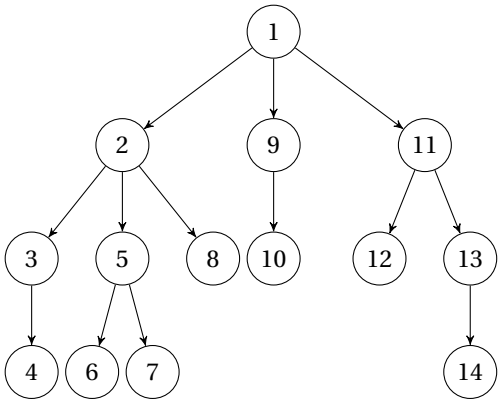


Figure 3.3: Depth-first search (DFS) algorithm illustration. The nodes are visited according to the order of the numbers.

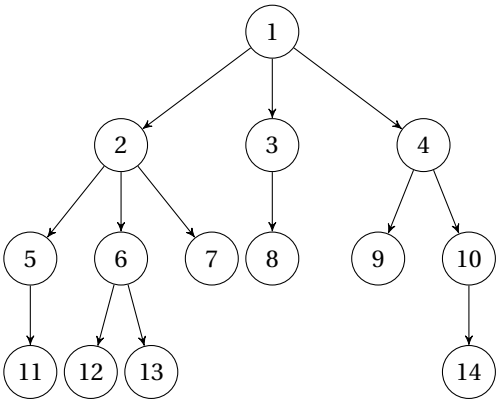


Figure 3.4: Breadth-first search (BFS) algorithm illustration. The nodes are visited according to the order of the numbers.

3.3. From routing classical to quantum data

3.3.1. Qubit topology

As qubit technologies evolve, different qubit topologies (or qubit arrangements) are appearing. These topologies can be classified into 1D, 2D, and 3D arrangements. The 2D arrangement of qubits seems to be the most promising topology [8], not only for short-term purposes but also for long-term purposes when FT quantum computing will be introduced and hundreds/thousands of qubits are used [31]. Therefore, this thesis will only focus on the routing of qubits on 2D qubit arrangements.

The current goal for a single quantum chip, e.g., Transmons quantum chip, is to arrange the qubits in a 2D lattice. The layout of this lattice is fixed, and only operations between direct neighbors can be performed. This current lattice, also known as qubit plane, has some limitations in the possible interactions and operations performed between different qubits. In the next enumeration, the essential constraints which are taken into account in this thesis are listed.

- Qubits are connected in a 2D lattice. Meaning the maximum number of connections between each qubit and other qubits is four. This is can be seen in Figure 3.5.
- Qubits located at the edge of a 2D lattice will have fewer connections.
- Qubits can only interact with their direct neighbors. This interaction means that only a qubit which is directly connected to another qubit can perform a two-qubit gate. In Figure 3.5 this can be seen as the black lines between the qubits. Only qubits connected via a line can perform a two-qubit gate.
- If a two-qubit gate needs to be performed on qubits which are non-NN, these two qubits needs to be brought physically next to each other first before the two-qubit gate can be performed. This transportation is done by exchanging states between qubits.
- At any moment in time only one gate can be operating on a qubit. This time restriction means that a qubit can only be used for one operation at a time.

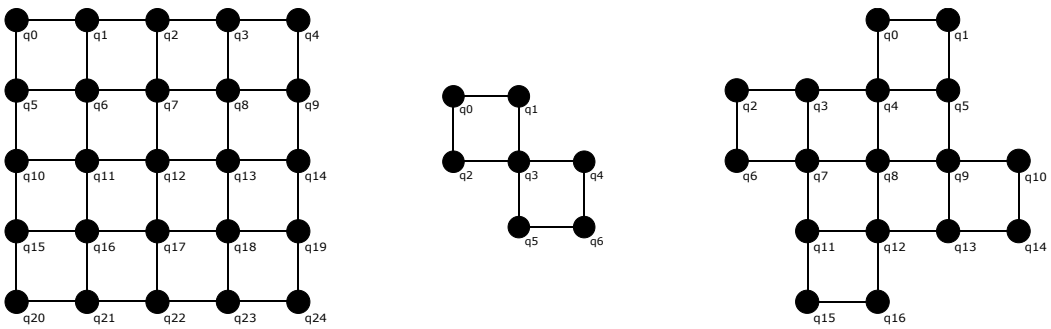


Figure 3.5: 2D lattices of physical qubits where the qubits are labeled. The 5x5 square lattice, SC-7, and SC-17 are illustrated from left to right.

3.3.2. Related work on routing of quantum states

Related work on how to route quantum states on different qubit topologies has been performed. Some of this related work will be reviewed in this section.

In [27], basic reordering of qubit positions in a 1D lattice has been evaluated. This reordering of qubit positions is based on finding the optimal order of qubits before performing gates.

In [5] [23] [32], different heuristic methods are used to minimize the number of resources when moving qubits around. In this case, the number of resources is the number of inserted movement operations, i.e., SWAP gates. A SWAP gate is a quantum gate where two qubit-states are interchanged (see Section 3.4). To minimize the number of SWAP gates, a path priority-calculation is designed. The path priority is fundamentally based on the initial instruction priority, the shortest path-length and the path with the lowest number of congestions; in which a qubit is blocked for routing. The instruction priority is defined by the various scheduling algorithms. The lowest number of congestions is calculated by the effect a certain path has on the future occupations of qubits. The shortest path is selected based on the lowest number of SWAP gates. When all the paths are inserted, the final quantum circuit is rescheduled. A small adjustment for classifying the number of congestions is made in [33], not by looking at the used qubits but at the number of SWAP gates that need to be performed in total. In this case, different paths may lead to a different total number of SWAP gates between all two-qubit gates.

For large quantum circuits, routing based on a sliding window or a subset of the circuit was introduced. The shortest path is based on the shortest path-length and the total minimum number of SWAP gates to route the remainder of the two-qubit gates inside the window. This routing is performed by the compiler, for 1D qubit lattices [28] and for 1D/2D qubit lattices [21] [34].

Several methods of path searching and/or sorting qubits are used to minimize the number of SWAP gates. These pathfinding algorithms are based on some of the classical algorithms introduced in chapter 3.2. Some routing algorithms try to optimize the entire quantum circuit at a time [22] [29], while some others optimize a subset of the quantum circuit or use a sliding window [1] [35]. Also, the effect of various lattice topologies has been tested using different routing algorithms, which select the shortest path for a single two-qubit gate [3].

From this related work on routing of quantum states, it is clear that finding the shortest path and looking at other two-qubit gates can have a positive effect on the total number of inserted SWAP gates. Also, the use of a sliding window and various pathfinding algorithms may be beneficial when mapping large quantum circuits. What is interesting to see is that the optimization in other literature is mainly done based the number of SWAP gates inserted in a quantum circuit. Because optimizing the number of added SWAPs may not lead to the lowest execution time of a quantum circuit. Therefore an optimization of the execution time of the quantum circuit may also be interesting since the total execution time is directly related to the coherence of the qubits.

3.4. Qubit transportation

Before introducing the algorithm for routing quantum states, first, different transportation techniques for moving quantum states need to be examined. Two variables are used to describe the different transportation techniques: the size, and the logic depth of the quantum circuit. The size of the quantum circuits is the total number of gates inside the quantum circuit. The logic depth (LD) of a quantum circuit refers to its execution time when each gate has a duration of 1.

3.4.1. Physical qubit transportation

SWAP-based transporation

The basic method for moving quantum states is the SWAP operation. The SWAP operation exchanges two qubit states [24]. This SWAP operation can be implemented by performing 3 CNOT gates on the two qubits as shown in Figure 2.4. If data needs to be transported over larger distances these SWAP operations can be repeated. The LD of the quantum circuit required for moving a qubit to a different location is $3 * n - 3$ (with

n the number of qubits across the path). The size of this quantum circuit is the same as the LD, $3 * n - 3$. An example can be found in Figure 3.6 in which $|q_0\rangle$ is moved to $|q_6\rangle$.

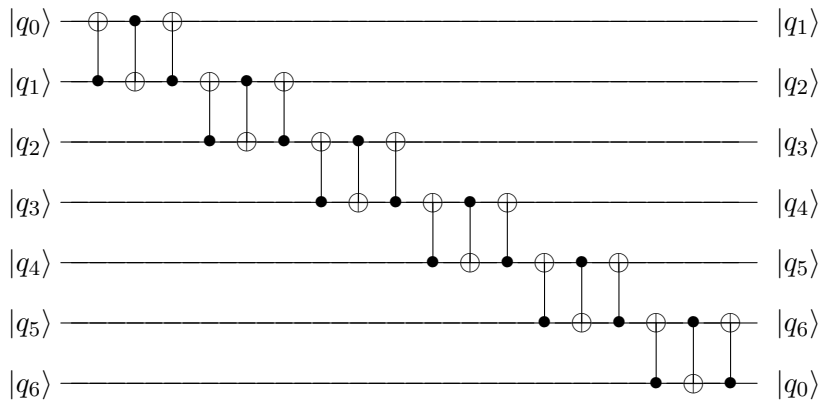


Figure 3.6: Qubit SWAP along multiple qubits.

To improve the LD as well as the quantum circuit size when SWAPs are used for moving qubits, different methods are proposed.

Rotation: When data needs to be sent from A to B the principle of rotation can be used as shown in Figure 3.7. The qubits are rotated one position. In this case, more gates are required, but the LD is decreased. The LD of this circuit is $n+5$, and the circuit has a size of $4 * n - 6$.

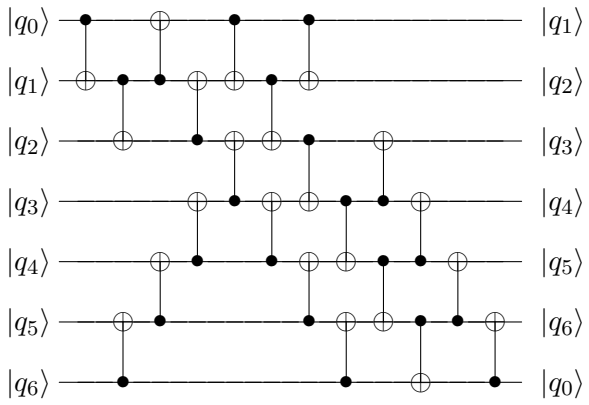


Figure 3.7: Transporting qubits by using the rotating operation [18].

SWAP with lower logical depth (LLD): The LD of the standard SWAP quantum circuit can be lowered by reordering the CNOT gates as shown in Figure 3.8. While doing so, the data is moved in both directions from A to B. As a result, the total execution time of this circuit decreases. The LD is $n+7$ for an even and $n+8$ for an odd number of qubits, while the original quantum circuit of Figure 3.6 has an LD of $3 * n - 3$. The size of the circuit is $6 * n - 9$.

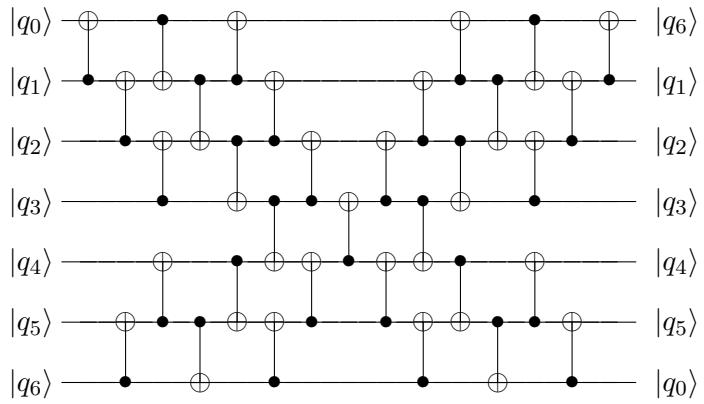


Figure 3.8: SWAP along multiple qubits with an LLD of $n+7$ for an even number of qubits, and $n+8$ for an odd number of qubits [18].

Reversal: For moving quantum states in both directions, $|q_0\rangle$ to $|q_6\rangle$ and $|q_6\rangle$ to $|q_0\rangle$, S. Kutin proposed another quantum circuit called reversal SWAP [18]. This quantum circuit can be seen in Figure 3.9. The qubits are mirrored in position as can be seen in the figure. This quantum circuit has an LD of $2*n+2$ and a size of n^2-1.

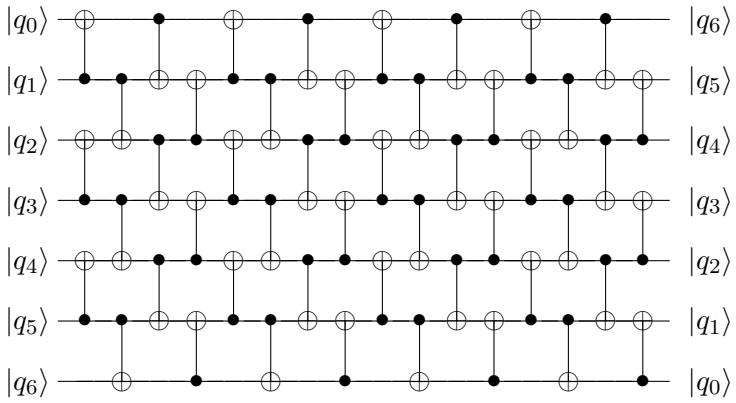


Figure 3.9: Transporting qubits by using the reversal operation [18].

SWAP with ancillas: The last method based on SWAP operation is the use of ancilla qubits when performing a SWAP operation. When these qubits are only used for transportation, these qubits are assumed to be preset to the zero state. Transportation along these ancilla qubits will only require two CNOT gates instead of three. This quantum circuit can be seen in Figure 3.10. It has an LD of $n+1$ and a size of $2*n-2$

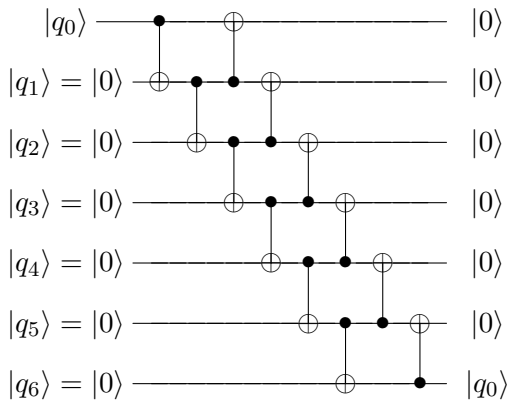


Figure 3.10: SWAP along multiple qubits by using ancilla qubits.

Teleportation-based transportation

Another method for moving qubits states is by using teleportation. Teleportation is performed by generating an EPR-pair on two qubits and transporting one or both qubits to different locations. The state from a third qubit can be transported to one of the qubits that form the EPR-pair by using the other qubit of that pair. The quantum circuit of teleportation circuit can be seen in Figure 3.11, the explanation of teleportation is done by referring to this figure. An EPR-pair is a pair of qubits that are in the Bell state (one of the four maximally entangled states). In Figure 3.11 an EPR-pair is generated with qubits $|q_1\rangle$ and $|q_2\rangle$ by performing an H-gate on $|q_1\rangle$ and a CNOT-gate with $|q_1\rangle$ and $|q_2\rangle$. Because of this entanglement, measurement on one of the qubits will force both qubits to fall in same basic state immediately. This property can be used to transport the state of a third qubit, $|q_0\rangle$, with the use of one of the qubits of the EPR-pair, $|q_1\rangle$, to the other qubit of the EPR pair, $|q_2\rangle$. This transportation is done by performing a so-called Bell measurement between the qubits $|q_1\rangle$ and $|q_0\rangle$, by performing a H-gate on $|q_0\rangle$, a CNOT-gate on $|q_0\rangle$ and $|q_1\rangle$ and to measure both qubits. The measurement results are sent through classical communication channels to the other qubit, $|q2\rangle$, that belonged to the EPR-pair to perform a controlled X- and Z-gate to complete the transportation.

The main disadvantages of teleportation is the area needed to generate an EPR-pair and the complexity to distill a high-fidelity, perfect, EPR-pair [4]. How this process is performed is beyond the scope of this thesis.

The main advantage of teleportation is that a qubit state can be sent over a long distance in a fraction of the time compared to the use of SWAP gates. If the EPR-pair is prepared and distributed before the actual transportation is needed, a lot of time can potentially be saved.

There are different methods for distributing EPR-pairs:

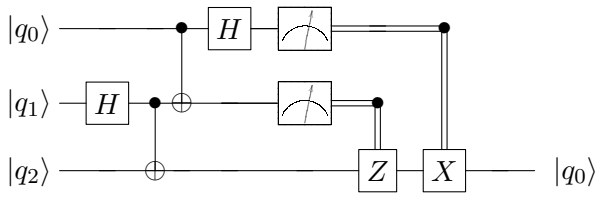


Figure 3.11: Quantum teleportation by generating an EPR-pair on qubit q1 and q2 (first H and CNOT gate) and performing a Bell measurement (second CNOT and H gate) on q0 and q1.

Centralized EPR-pair distribution: To distribute the qubits of an EPR-pair, all the methods introduced for the SWAP gate can be used. Methods for continuous generation of multiple EPR-pairs, where EPR-pairs are constantly moved away from the origin, as well as single EPR-pair distribution, can be used. An example of this quantum circuit can be found in Figure 3.12. In this figure, normal SWAP operations are used to distribute the EPR-pair.

The advantage of this method is that after the EPR-pair is moved to the next qubit, a new EPR-pair can be generated on the centralized qubits. Therefore a channel filled with EPR-pairs can be formed.

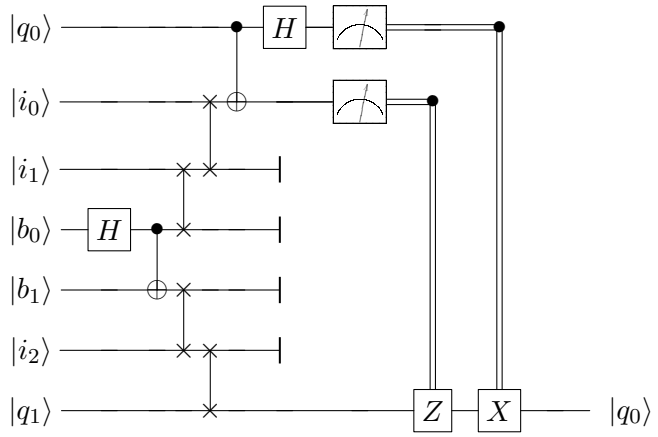


Figure 3.12: Distribution of an EPR-pair using SWAP operations.

Decentralized EPR-pair distribution: By generating EPR-pairs along multiple qubits instead of using SWAP gates to distribute the EPR-pair, the logical depth can be decreased a lot. To do so, multiple EPR-pairs are generated on a single row of qubits. The intermediate steps are measured in the X- and Z-basis. The corrections are performed on one of the end qubits to form one single EPR-pair. The quantum circuit for this option can be found in Figure 3.13.

By using this principle, multiple EPR-pairs can be combined to form a quantum circuit such as in Figure 3.14. In this situation, qubit $|q_0\rangle$ is moved using teleportation to qubit $|q_6\rangle$. Because multiple measurements will tell if a Pauli-X and/or -Z gate needs to be performed, this quantum circuit can be simplified and executed with the same duration as can be seen in Figure 3.15. The wire intersections in this figure need to be replaced with classical digital logic. This logic will perform the Pauli gate only if an odd number of positive measurement outcomes has occurred.

The major disadvantage of using decentralized EPR-pairs is that the entire channel is blocked for any other operation. Also, only one EPR-pair can be distributed at a time.

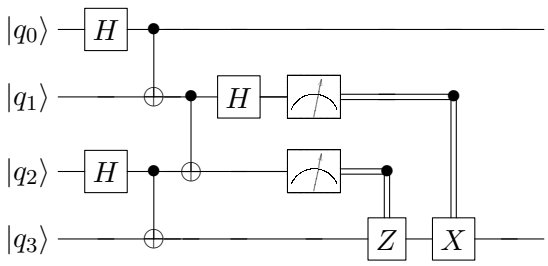


Figure 3.13: Combining two EPR-pairs to form one EPR-pair with a larger distance.

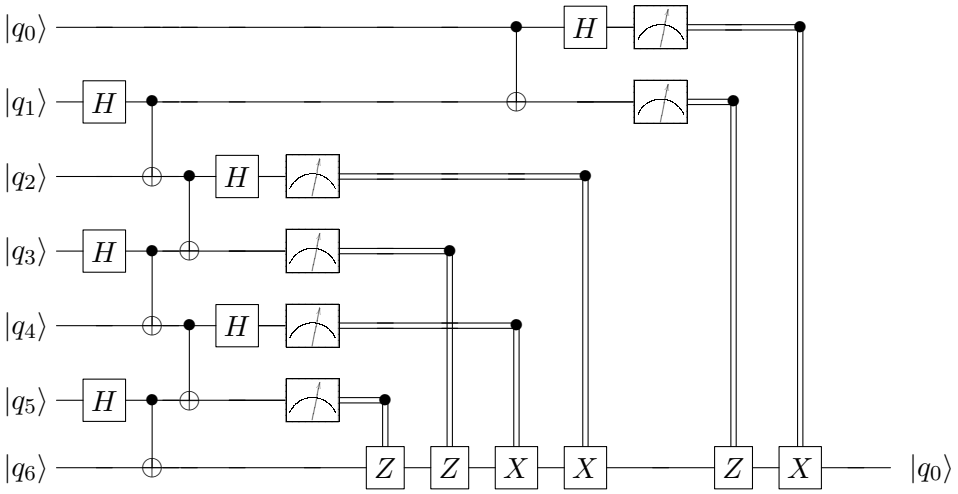


Figure 3.14: Distribution of an EPR-pair using multiple EPR-pairs.

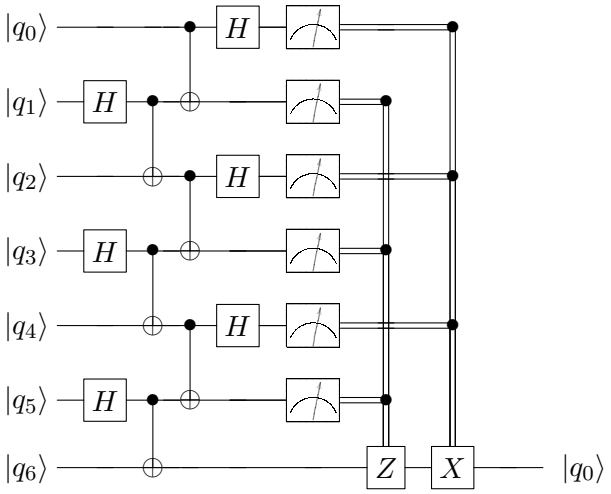


Figure 3.15: Distribution of an EPR-pair using multiple EPR-pairs.

Increase fidelity of EPR-pairs

When creating channels with EPR-pairs, an EPR generator can be located in the middle. When the EPR-pairs are transported to the end of the channels, the fidelity of these EPR-pairs will be lower. The longer the channel, the lower the fidelity of the transported EPR-pairs. By using a specialized Teleportation Unit (see Figure 3.16) the state of the EPR-pair can be purified. When the EPR-pair state is purified the fidelity increases [4].

By using this method, a constant flow of EPR-pairs can be generated which might have advantages in throughput compared to the other options.

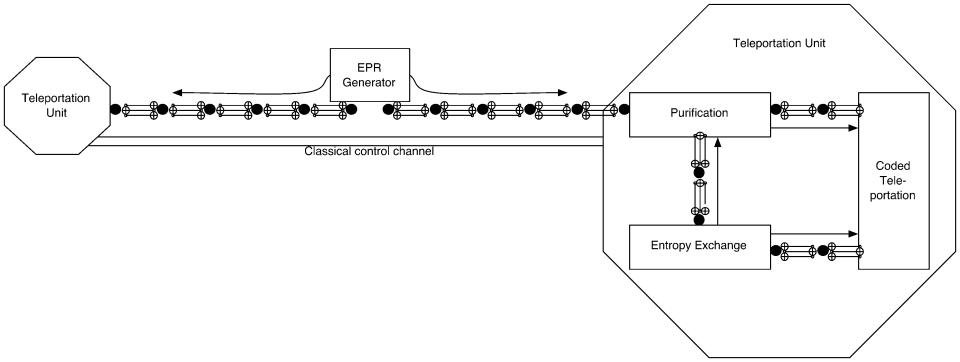


Figure 3.16: Example of a teleportation channel with an EPR-generator [4].

3.4.2. Logical qubit transportation

The transportation methods discussed in Subsection 3.4.1 can be also used for moving planar-based logical qubits. However, to perform a two-qubit gate (CNOT gate) on planar-based logical qubits, lattice surgery is needed (see Section 2.2). As explained before, to perform a SWAP operation, three CNOT gates should be performed between the control and target logical qubit (see Figure 2.4). This SWAP operation can be seen in the quantum circuit of Figure 3.17. The first CNOT operation is performed in the same way as in Section 3.4.1. In the next CNOT gate, the function of the target and control logical qubit is switched. Therefore the same ancilla can be used to do a rough merge and split first with the new control 'T'. To complete the second CNOT gate operation a final rough merge and split are performed with the new target 'C'. The last CNOT is again based on the first CNOT to complete the SWAP operation. The principle of the SWAP operation is the same when performing a SWAP operation on a planar-based SC-17 logical qubit.

To use teleportation on logical qubits, also the standard methods could be used from Subsection 3.4.1. The methods to distill a high fidelity EPR-pair is a complex and extensive process and not interesting for a grid with a small number of logical qubits. Therefore logical qubit teleportation will not be further discussed in this thesis.

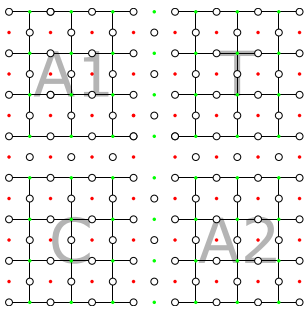


Figure 3.17: SWAP operation of a planar-based qubit through lattice surgery. The control, "C", and target, "T", interact with each other by merging and splitting with the intermediate surfaces, "A1" or "A2".

3.4.3. Evaluating the methods of transportation

For all methods of transportation, a series of operations need to be performed. These operations need to have a high fidelity to make sure the right state is transported. Reducing the number of gates and the total execution time required for moving quantum states will help to reduce the failure rate of computation. To find a suitable qubit state transportation method, the various methods are evaluated with respect to the number of gates required (circuit size), and the circuit execution time. To do so, a Python script is written.

Circuit size: To be able to asses the number of gates, the assumption is made that all different types of operation will have weight '1'. This same weight means that a measurement operation has the same weight as a CNOT operation. The total number of gates depends on the distance and the method of transportation.

Execution time: The execution time of a circuit cannot be assigned with the assumption that all operation will have the same weight. Therefore the LD, used for evaluating the different SWAP operations, cannot be used. A measurement operation has a longer execution time compared to for example a CNOT gate. Therefore the Python script is parameterized with a different execution time for each gate. The used execution time used for this script for each gate can be found in Table 3.1. These execution times are based on superconducting qubits [31].

Table 3.1: Weight for the different gates used to evaluate the various transportation methods [31].

Gate	Time [ns]
Classically controlled Pauli-gate	20
H	40
CNOT	80
Measurement	500

SWAP-based transportation

With these weights from Table 3.1, first the transport operations based on normal SWAP operations can be evaluated. In Figure 3.19 the simulation results for the various SWAP operations at different distances have been plotted. In the left graph, the number of gates versus the distance in qubits has been plotted. In the right graph, the logic depth and the total execution time is set out against the distance in qubits.

From these figures, it can be seen that the SWAP operation with the use of ancilla qubits is always beneficial with respect to the number of gates and the logic depth. In this case dedicated ancilla channels will be required for communication. When no dedicated communication channels are predefined, the SWAP based on rotation is the best option with respect to execution time, is the fastest transportation option inside all qubit lattices but uses more gates compared to the standard SWAP operation.

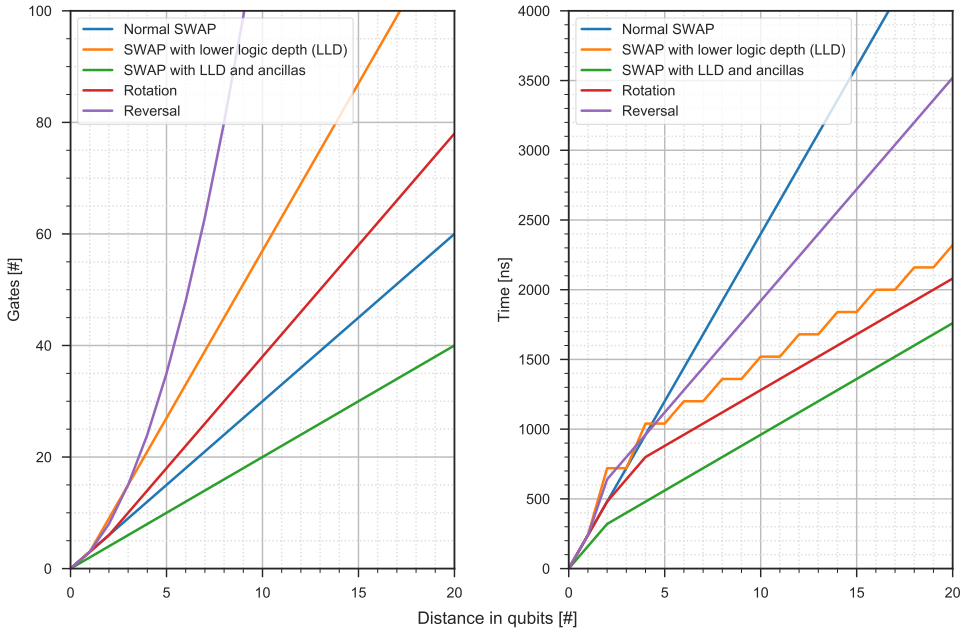


Figure 3.18: Number of gates with their corresponding logic depth and total execution time for different swap operations at different distances.

Teleportation-based transportation

Next, the transportation operations based on teleportation are simulated. For every option, a single EPR-pair is generated (except for the decentralized EPR-pair version) and spread using the different methods of the SWAP operation. The results can be found in Figure 3.18. The Python script is designed in a way that the EPR-pair generation can be done at any location in the line of connected qubits. In these simulation results, the EPR-pair is generated in the center of the line.

From Figure 3.19 it can be seen that, depending on the distance, different transportation methods might be preferred. The strength of the decentralized EPR-pair is the higher fidelity of the qubits at both locations. The major disadvantage of this method is the fact that the stream of produced EPR-pairs is discontinuous. An entire channel is blocked to produce one EPR-pair at both ends of the channel. The stream of EPR-pairs is an advantage for the normal SWAP and SWAP with lower logical depth where a continuous flow of these pairs can be distributed. The produced EPR-pairs are directly moved to the neighboring qubits and give the possibility to generate a new EPR-pair at the same location. Depending on the number of EPR-pairs needed and the distance, different methods are useful.

Another method could be to pre-generate and distribute the EPR-pair before the actual transportation is needed. In Figure 3.19 this pre-generated and distributed EPR-pair execution is represented by the solid light blue line. The execution time is for every distance the same because only a measurement and correction needs to be performed. What is not visible for this execution time is the time an EPR-pair is generated and distributed in advance to different locations.

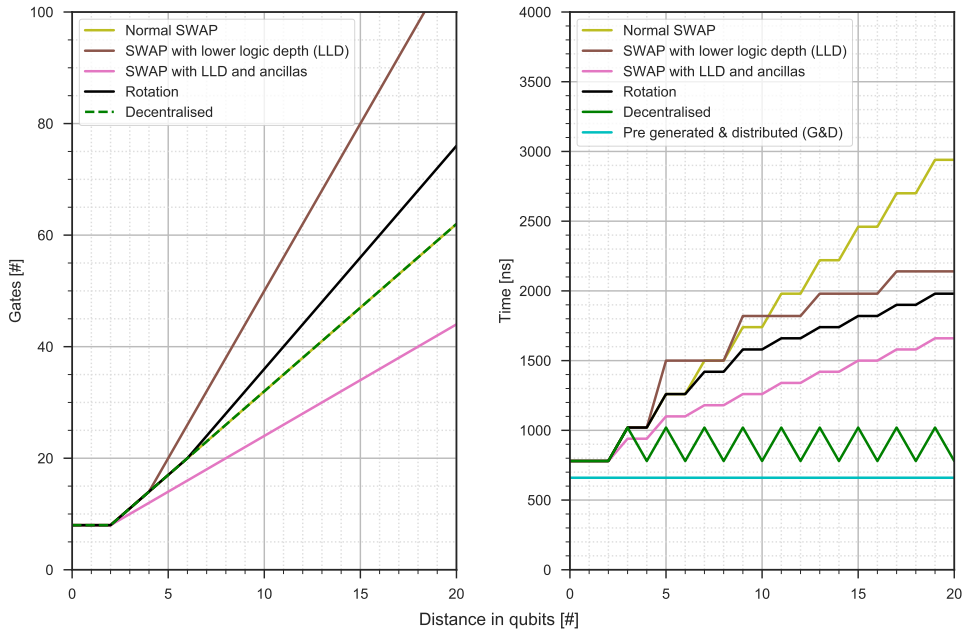


Figure 3.19: Number of gates with their corresponding total execution time for different teleportation distribution operations at different distances.

Conclusion

In Figure 3.20 the execution times for all the transportation methods is plotted. Transportation based on teleportation starts to be interesting already from a distance of four qubits when only taking execution time into account. The major challenge is to get the fidelity of the EPR-pairs high. This fidelity challenge as well as the added overhead of classical control to complete the transportation, are both not visible in the graphs of this chapter. Due to the small lattice sizes, up to 49 qubits, the next chapters will only make use of the SWAP operation. Only the standard SWAP will be used in the routing algorithm. The standard SWAP is easy to implement, and it is the fastest alternative concerning latency for these small distances. The SWAP with ancilla qubits is not used because the qubit planes used are minimal square lattices, i.e., the total number of qubits equals the number of qubits needed in the circuit. Therefore no dedicated ancilla qubits can be specified. Also, the required $|0\rangle$ state of these qubits introduces extra overhead what makes the SWAP with ancilla qubit not useful in these small lattice sizes. The rotational SWAP will not be used due to the added number of gates. Also, this rotational SWAP occupies more qubits from the beginning of the transportation, see Figure 3.7, which could potentially block the possible interleaving with other gates.

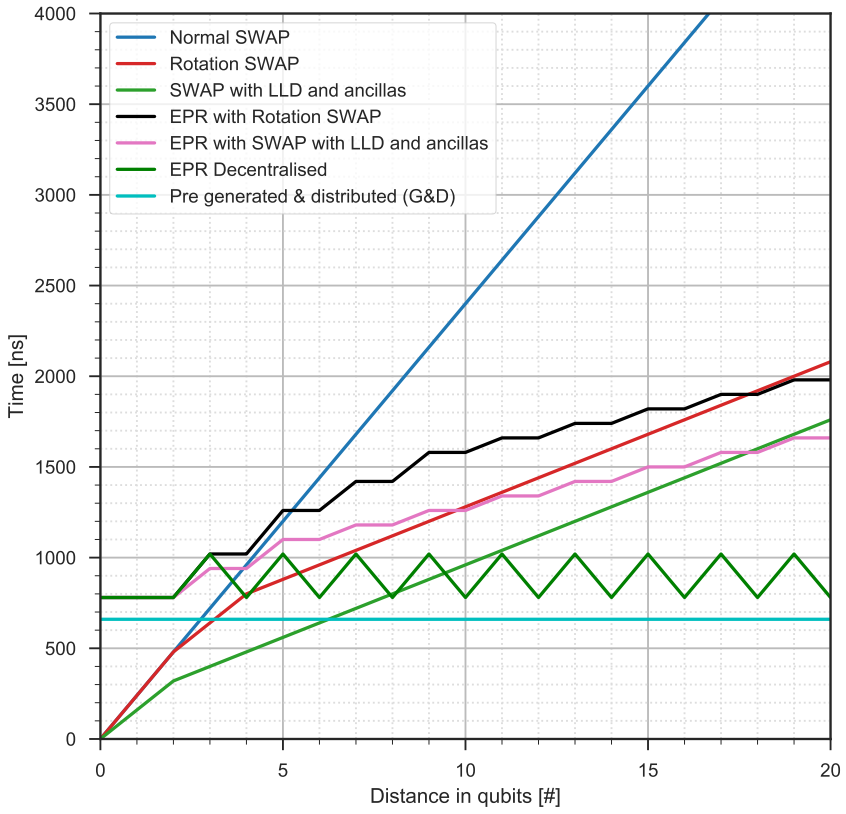


Figure 3.20: Total execution time for the various transportation methods for different distances.

4

Routing quantum states

Because of the nearest neighbor constraint, additional operations for routing qubits are required. This routing will most likely increase the execution time and then the failure rate of the computation. Therefore, an efficient routing algorithm which minimizes the communication overhead will help to improve the computation fidelity. In Section 4.1, the Quantum Assembly Language (QASM) used to describe quantum circuits is introduced. Also is explained how the compiler schedules such QASM-based instructions. These scheduled QASM-based operations will be the input of the routing algorithm proposed in Section 4.2. How the proposed routing algorithm is adjusted to be used for routing planar-based logical qubits is explained in Section 4.3. Finally, in Section 4.4 the metrics used to evaluate the performance of the routing algorithm is described.

4.1. Describing quantum circuits: QASM

The input of the routing algorithm proposed in this thesis will be a file in which a quantum circuit is described in QASM. In this section, it is explained how such a file looks like as well as how it is generated by the compiler.

To describe a quantum circuit (or quantum algorithm), a standard language is needed. This language should be able to specify the quantum gates with the corresponding qubits. To do so the Quantum Assembly language (QASM) has been developed, which includes classical as well as quantum instructions. QASM-based instructions describing the quantum circuit are generated by the compiler as illustrated in Figure 4.1. Those instructions are technology-independent and can be simulated on a quantum simulator, e.g., QX Simulator [15] or on different quantum processors.

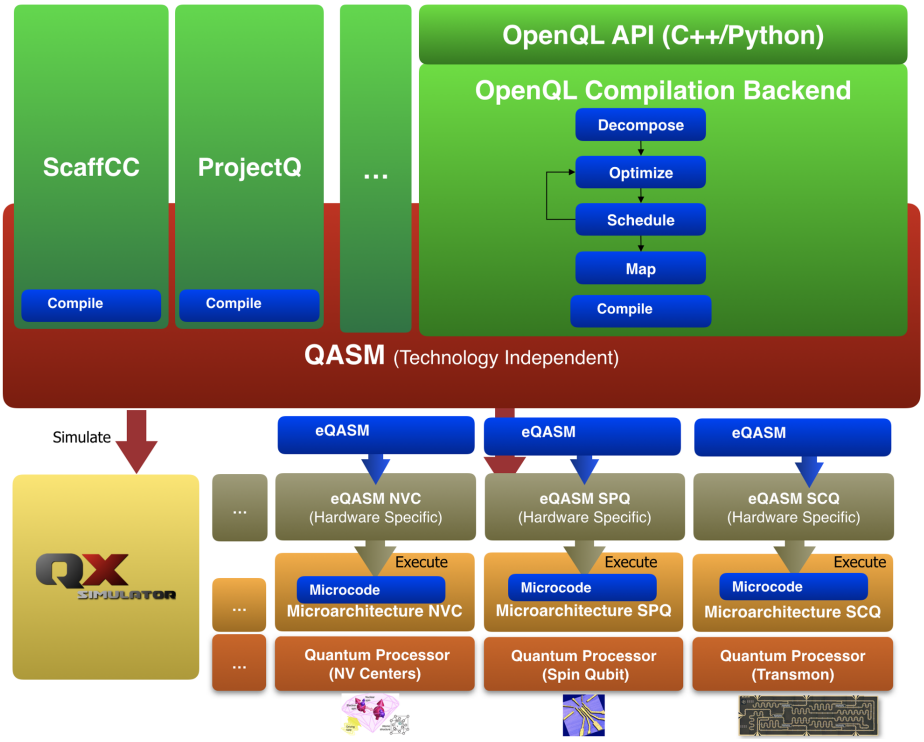


Figure 4.1: The quantum compilation stack. QASM forms the technology-independent language between the various compilers and the quantum simulation platform QX as well as the various hardware implementations [14].

In the QASM language, qubits are defined, and gates are specified to execute at a particular time. QASM is specified in the unpublished paper [14]. QASM defines how the gates are performed sequentially and in parallel. Gates on one instruction line, separated by a '|', are performed in parallel. The subsequent instruction lines are performed sequentially. Each instruction line will, therefore, have a duration of 1 time unit, in this thesis called cycle. In Figure 4.2 on the left, an example of a quantum circuit written in QASM is shown. This quantum circuit leads to the occupation of qubits as shown in the right of the image. In the example, no latencies are taken into account.

Line [#]	Instruction
1	CNOT q1,q0 | X q2
2	H q0 | CNOT q1,q2
3	Z q1

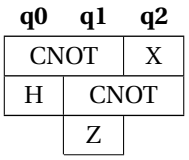


Figure 4.2: Left a quantum circuit written in QASM. This circuit occupies qubits as shown on the right. No latencies are taken into account.

To schedule a quantum circuit, the gate durations have to be taken into account. Also, it may be the case that in a cycle no instruction can be started. To describe this principle, QASM introduces the 'qwait' statement. During this qwait statement, no new instructions are fetched. The number of cycles in which no new instruction are fetched can be specified in the qwait statement. For example, 'qwait 3' will not fetch any gates for three cycles. When assuming different gate times (in number of cycles), the example presented in Figure 4.2 will translate to the QASM file shown in Figure 4.3. This QASM leads to the occupation of qubits as shown in the right of the image.

Line [#]	Instruction
1	CNOT q1,q0 | X q2
2	qwait 3
3	H q0 | CNOT q1,q2
4	qwait 2
5	Z q1

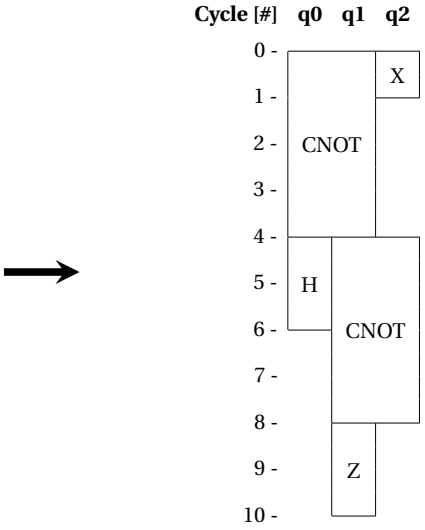


Figure 4.3: Left a quantum circuit written in QASM with the introduced qwait statement. This circuit occupies qubits as shown on the right. In this situation, the out-latency of each qubit of a multi-qubit gate is set to the highest out-latency.

The single-qubit gates inside a quantum circuit will only have a single duration (out-latency) while two-qubit gates can have a so-called in- and out-latency differing per qubit (how long each qubit is occupied). This in- and out-latency is a wait state for one of the two qubits. While a total latency of a two-qubit gate is the longest latency of both qubits of the gate, the individual qubits can perform another gate in parallel. The most common example of a two-qubit gate with different in- and out-latencies is the CNOT gate. The reason for different latencies for both qubits in a CNOT gate is that a CNOT gate can be decomposed into an H, CZ and H gate when executed, for instance, on superconducting qubits. Figure 4.4 shows this decomposition and how this affects the in- and out-latency of both qubits inside a CNOT gate. Note that in this case, the H-gate only has a duration of 1 cycle. However, in Table 4.1 a H-gate duration of 2 cycles is specified. This shorter duration has to do with the fact that an H-gate can be even decomposed further into a 90 degree Y gate and an ordinary X gate. In the combination of the CZ and the two H-gates, the number of gates can be optimized and be executed two cycles faster (4 cycles instead of 6).

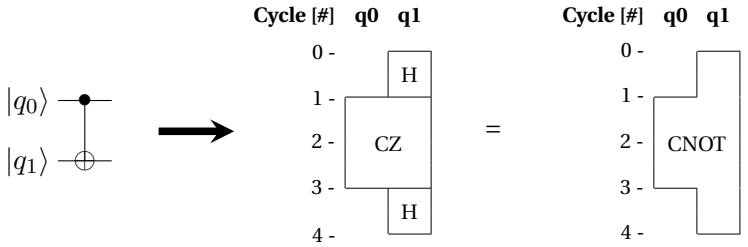


Figure 4.4: The in- and out-latency of a CNOT gate. The different in- and out-latencies of the qubits in a CNOT gate is there because a CNOT gate when executed on superconducting qubits can be decomposed into an H, CZ and H gate, and the control qubit only takes part in the CZ gate.

The use of in- and out-latencies, of for example the qubits of a CNOT gate, allows interleaving of gates as shown in Figure 4.5. Note that the final execution time (7 cycles) will be lower compared to the non-interleaved version (10 cycles). The same holds for a SWAP operation on two qubits. This SWAP operation can be decomposed in three CNOT gates as shown in Figure 4.6. The execution time will be 10 cycles instead of the three times the latency of a CNOT gate which will result in an execution time of 12 cycles.

Line [#]	Instruction
1	CNOT q1,q0 | X q2
2	qwait
3	CNOT q1,q2
4	qwait
5	H q0
6	Z q1

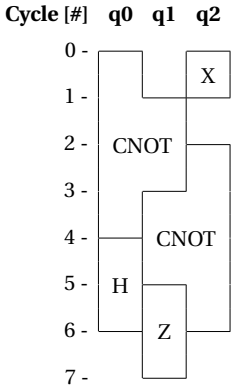


Figure 4.5: Left a quantum circuit written in QASM with the introduced qwait statement. This circuit occupies qubits as shown on the right. In this situation, the actual in- and out-latencies of all qubits of all gates is used.

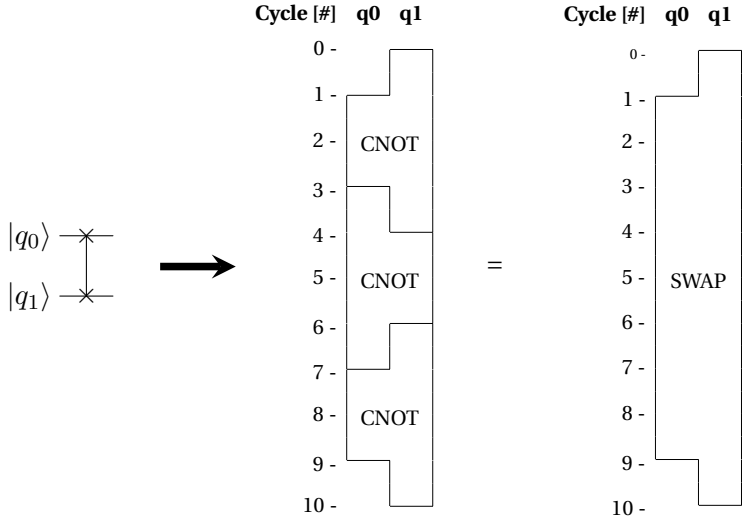


Figure 4.6: The in- and out-latency of a SWAP gate. The different in- and out-latencies of the qubits in a SWAP gate is there because a SWAP gate can be decomposed into three CNOT gates which can interleave.

4.2. The proposed routing algorithm

In algorithm 1 the routing algorithm proposed in this thesis is given. This algorithm is a summary of the entire routing logic. The inputs are a scheduled QASM-file generated by the compiler, the qubit topology, and the initial placement of the qubits. The output is a routed QASM-file in which the required move operations have been inserted to satisfy the NN constraint. How the routing algorithm works is also shown in Figure 4.7 in the form of a block diagram. In this block diagram, the input QASM-file is loaded and checked whether the qubits involved in a two-qubit gate are NN. If this is not the case, the routing algorithm will find and select a path based on certain criteria explained in the next subsections. After the path has been selected, the SWAP gates are inserted, the instructions are rescheduled till the original non-NN gate, and the first instruction cycle is written to an output file. These steps continue until the entire file has been processed. In the resulting output QASM-file, the NN constraint is satisfied.

The different parts and functions of the proposed routing algorithm are explained in the next subsections.

Algorithm 1 Routing algorithm

Input: Defined qubit topology and its size (lattice), initial placement, scheduled QASM-file
Output: Routed QASM-file

```
Define instruction buffer B, length l = window size [#cycles]
Fill B with instructions from input-QASM
while B is not empty do
    # Check if an instruction (ins) is NN
    for ins in B[0 : l/2] do
        # If an instruction is non-NN, start routing
        if ins is not NN then
            # Split parallel instructions (pp_op)
            B, insert empty cycle before and after ins line.
            if pp_op == NN then
                pp_op.shift(one cycle earlier)
            else pp_op.shift(one cycle later)
            # Find paths
            paths = all shortest paths for ins based on BFS
            # Split different paths allowing movement of control and target
            paths+ = all intermediate locations for each path
            # Different transportation techniques, standard SWAP based
            for p in paths do
                p.length = #cycles from p
            # Look-back
            for p in paths do
                p.length − = #cycles p can interleave with instruction in B[0 : ins]
            # Look-ahead to other ins (o_i)
            for p in paths do
                Place qubits based on p
                p.length += ∑_{o_i∈B[ins:l]} shortest path length of o_i in #cycles to comply to the NN contraint
            Insert path with min. length in buffer and update placement
            Break for-loop
    Reschedule B[0 : ins]
    Write B[0] to output-QASM and remove from buffer
    while B.length < l do
        Fill B with new instruction cycle from input-QASM and apply qubit placement on it
```

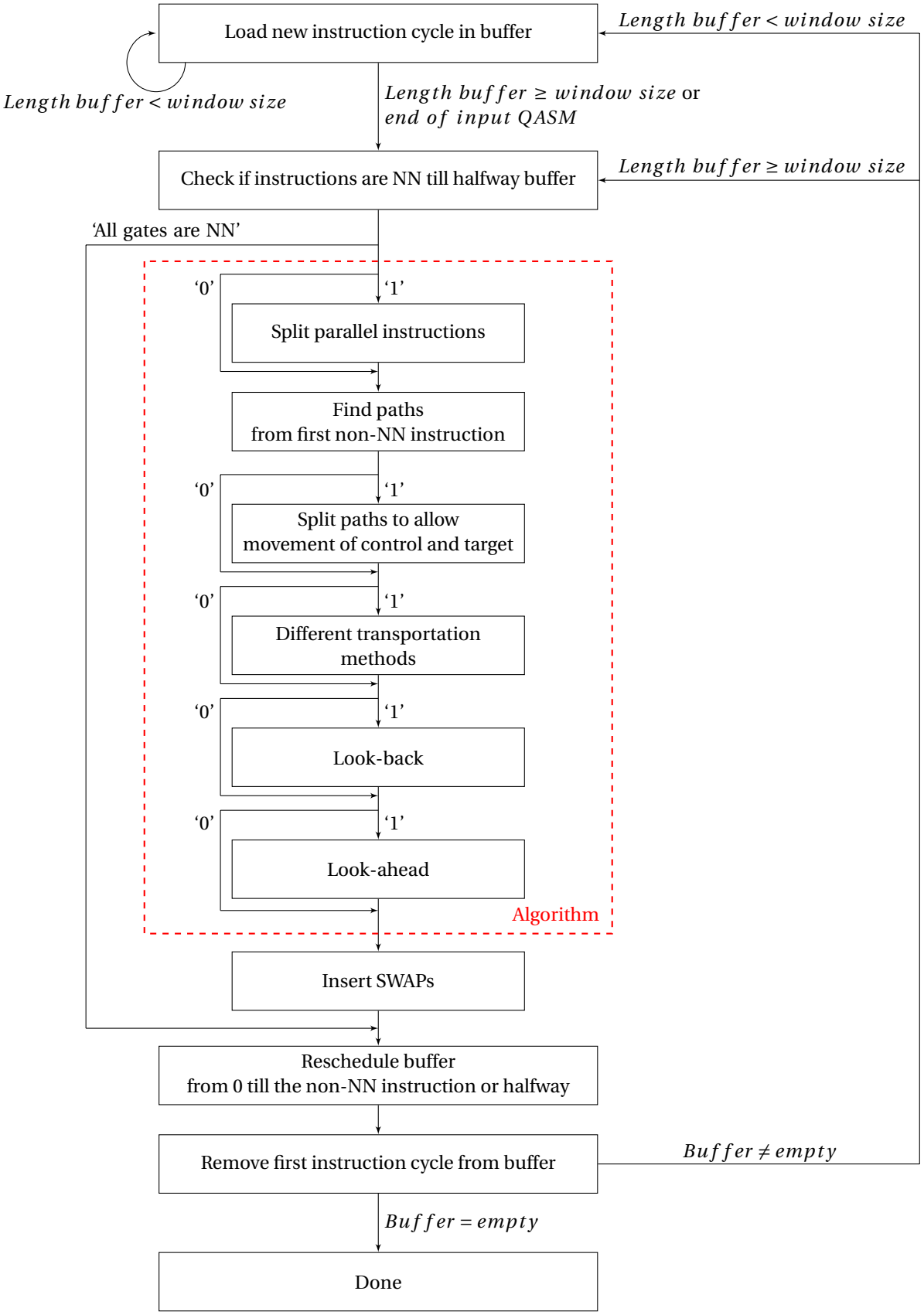


Figure 4.7: Block diagram of the total routing algorithm including the in- and output. The '0' and '1' indicate wheter a function is disabled or enabled respectively.

4.2.1. Input/Output and sliding window

The routing algorithm requires two parts as input: a pre-scheduled QASM file and a qubit lattice topology. The pre-scheduled QASM file contains a quantum circuit which is scheduled and obeys the rules concerning the duration of gates. The qubit lattice topology is a pre-defined structure of connected qubits in a 2D plane. Any lattice shape can be defined as long as the qubits are arranged in a 2D array. With those two components, the qubits defined in the QASM file are then routed based on the restrictions of the qubit lattice.

The QASM file will be read line by line. Each line of parallel instructions is fetched in a buffer. This buffer expands while adding instruction cycles. Initially the buffer is filled till it has a length equal to the window size. The window size is expressed by a number of cycles. Only instructions inside this buffer will be used to look-back and look-ahead, functions that will be explained later. During the execution of the routing algorithm, another buffer is used for rescheduling the instructions. In the following list, the buffers are explained.

- Buffer1: In this buffer, all instructions are stored as an exact copy of the QASM file. Only the parallel components are split. Qwait instructions are stored as 'X' number of empty rows, where 'X' corresponds to the number of empty cycles where no new instructions are fetched. An example based on the quantum circuit of Figure 4.5 can be found in Figure 4.8.

- Buffer2: In this buffer, each physical qubit is represented by a single column, and each row again represents a cycle. In each instruction slot, a cell will be filled with an 'X' if the qubit is occupied by a single-qubit gate, a 'Y' when occupied by a two-qubit gate, or left empty if not. When a two-qubit gate instruction is fetched in this buffer, for the first occupied qubit due to the two-qubit gate, the 'Y' will be replaced by the number of the corresponding qubit on which the two-qubit gate acts. This number will visualize the connectivity between qubits needed for moving gates inside the buffer. In Figure 4.8 the relationship between buffer1 and buffer2 is illustrated.

Line [#]	**Instructions**
1	CNOT q1,q0 | X q2
2	qwait
3	CNOT q1,q2
4	qwait
5	H q0
6	Z q1
7	...

	Buffer1		**Buffer2**		
Cycle [#]			**q0**	**q1**	**q2**
1	CNOT,1,0	X,2	1		X
2			Y	Y	
3	CNOT,1,2		Y	Y	1
4			Y	Y	Y
5	H,0		X	Y	Y
6	Z,1		X	X	Y
7				X	

Figure 4.8: The relation between the QASM from Figure 4.5 and the buffers used in the routing algorithm.

The qubit lattice will define which qubits are neighbors. To define the qubit lattice an $n * m$ square is configurable. From this lattice, qubits can be left out to form shapes or to simulate defects in this lattice. For mapping the virtual qubits (qubits in the QASM circuit description) to the corresponding physical qubits defined in the input qubit lattice and keep track of their location, the following two look-up tables are updated during the routing algorithm.

- Map1: In this list, the mapping from qubit names to the virtual level is stored as illustrated in Figure 4.9. This list is a simple look-up table which translates the qubit names in the QASM file, the virtual qubits, to a standard qubit name. This look-up table is fixed for the entire quantum circuit, and it is determined solely by the qubit lattice.

Physical qubit	0	1	2	3	4
Label in QASM file	q_0	q_3	q_4	q_2	q_1

Figure 4.9: Illustration of map1 used in the routing algorithm. The numbers corresponds to the physical qubits and the labels to the qubit names in the QASM file.

- Map2: In this list, the mapping of the virtual to physical qubit is stored as illustrated in Figure 4.10. This list has the same number of rows as buffer1. When two qubit states are interchanged at a certain moment of time, this mapping is also adjusted at the same cycle/row, this is illustrated in Figure 4.10 by the two SWAP operations. This mapping is then used to go from the to virtual QASM level qubit level to the physical level when new instructions are fetched in the buffers.

Line [#]	Instructions
1	
2	
3	SWAP q0,q2
4	
5	SWAP q1,q2
6	

Cycle [#]	q0	q1	q2
1	0	1	2
2	0	1	2
3	**2**	1	**0**
4	2	1	0
5	2	**0**	**1**
6	2	0	1

Figure 4.10: The relation between a SWAP operation and map2 used in the routing algorithm. No gate latencies are taken into account.

After running one cycle of the routing algorithm, the first line of the buffers is written to a new QASM file. The first line is also removed from all buffers. If the next line contains no instructions due to the out-latency of previous cycles all these 'empty' instruction lines are combined to form one qwait instruction.

4.2.2. Routing algorithm functionality

After buffer1 has been filled with instructions of which the length that equals the window size, the routing procedure starts. 1) The first part is to check whether the two qubits that are part of a two-qubit instruction are neighbors. If this is the case, the instruction with its corresponding qubits will be left untouched. If not, a path needs to be found between those two qubits. 2) The parallel instructions that obey the NN constraint are shifted to an earlier cycle to avoid possible extra transportation. 3) Different paths are found by using the BFS algorithm explained in 3.2.4. 4) In order to select the final path, each path is evaluated by how much the operations in those paths can be interleaved with earlier operations (look-back function) that the increased latency is minimized. However, the selected path will also have a negative or positive effect on the other later two-qubit gates; that is, the qubits involved in later two-qubit gates can be brought closer or further apart from each other due to a selected path (look-ahead function). The interleaving with earlier gates and the negative or positive effects on qubits that need to interact in the future are all translated in latency [#cycles]. This latency is checked and updated for all path options. 5) Finally, the fastest path option, with respect to latency in cycles, is selected and inserted in the buffers. In this section, the steps in finding an optimal path are explained in more detail.

Split parallel instructions

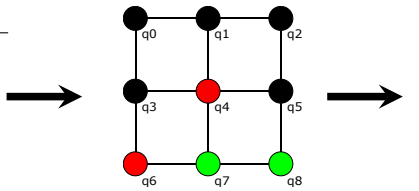
In a quantum circuit, gates can be executed in parallel. When an instruction does not satisfy the NN constraint, the parallel instructions which do satisfy the NN constraint and the single-qubit gates should be executed first. These instructions have the same priority concerning the entire dependency graph of the quantum circuit, therefore all these instructions have the same priority to make the qubits NN. There is the possibility that when a path is inserted, the original parallel NN gates are delayed and/or made non-NN due to the reordering of qubits. Both possibilities could add extra latency to the quantum circuit.

Figure 4.11 shows an simple illustration, how the principle of splitting parallel instructions works. In this quantum circuit, there are two CNOT gates executed in parallel. The qubits in the first CNOT gate are non-NN while the qubits in the second gate are NN. When a SWAP is inserted between q6 and q7, to make the first qubits inside the CNOT gate NN, the qubits in the second gate are moved away from each other. To avoid this possible extra overhead, the parallel gate with NN qubits will be executed first as shown in the right of Figure 4.11.

The split parallel instruction function is implemented by shifting, the parallel instructions that obey the NN constraint, to an earlier cycle inside the buffer. This splitting is done by first adding an extra cycle in

buffer1, and then shift the respective instructions that obey the NN constraint as well as the single qubit gates one cycle earlier. When this step is not executed the quantum circuit will still be completely routed, but there could potentially be extra latency added to the quantum circuit.

Line [#]	Instructions
1	X q0
2	CNOT q4,q6 | CNOT q7,q8
3	Z q5
4	qwait 2
5	...



Line [#]	Instructions
1	X q0
2	CNOT q7,q8
3	CNOT q4,q6
4	Z q5
5	qwait 2
6	...

Figure 4.11: A simple example of the effect of the splitting parallel instructions function. When the parallel CNOT gates are not split, and a SWAP operation is inserted between q6 and q7 to route the qubits in the CNOT q4,q6. The original CNOT q7,q8 operation is changed to CNOT q6,q8. Now this CNOT q6,q8 instruction is now non-NN, and extra routing steps are needed. This could have been avoided if this CNOT gate could be executed first as shown in the table in the right.

Finding paths

To find all the possible paths between two qubits, the breadth-first search (BFS) algorithm is used as explained in Subsection 3.2.4. In Figures 4.12 through 4.15 an example of this BFS algorithm is illustrated. In this example, the BFS algorithm will find a path from qubit eleven to qubit nine, see Figure 4.12. The BFS algorithm starts from qubit eleven (the control) and checks all its neighbors one by one; in Figure 4.13 this is illustrated by the blue qubits q7, q12, and q15. Next, it will look for its destination at the neighbors of q7 (see Figure 4.14), namely q6, q3, and q8. Next, it will jump to q12, etcetera until the BFS algorithm finds its destination.

The original BFS algorithm will stop after it finds the first path to the target. Therefore the BFS algorithm has been adjusted to not stop at the first path, but it will end after all possible paths at the same path length have been evaluated. In Figure 4.16 the final BFS algorithm graph is shown. In this graph can be seen that the BFS algorithm will not stop when it reaches its target but will continue searching if there are more paths at the same depth.

To explore even more options, the routing algorithm is also equipped with a variable called extra path length. The BFS algorithm will continue searching for more paths until the maximum path length is the shortest path plus the extra path length.

Moving only one of the two qubits (the control and target) may not be the optimal solution. Therefore each of the possible paths is split into all the options of moving both qubits to any intermediate step. These path options are stored in a list with paths with their total execution time.

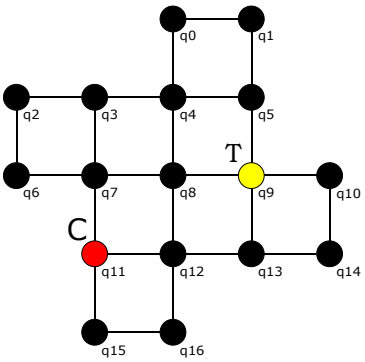


Figure 4.12: Finding possible paths inside a lattice from q11 to q9.

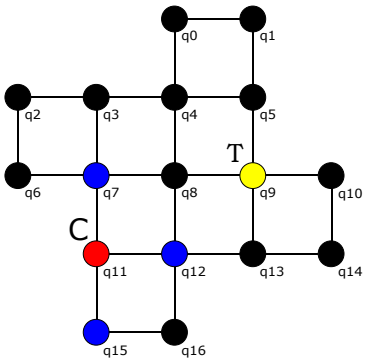


Figure 4.13: BFS step 1.

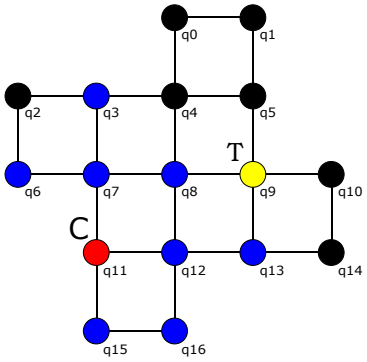


Figure 4.14: BFS step 2.

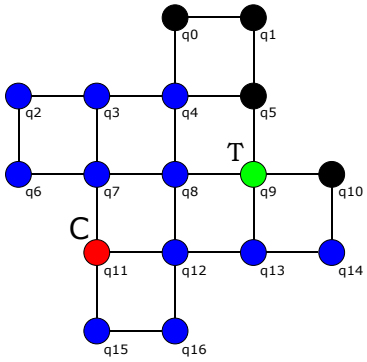


Figure 4.15: BFS step 3.

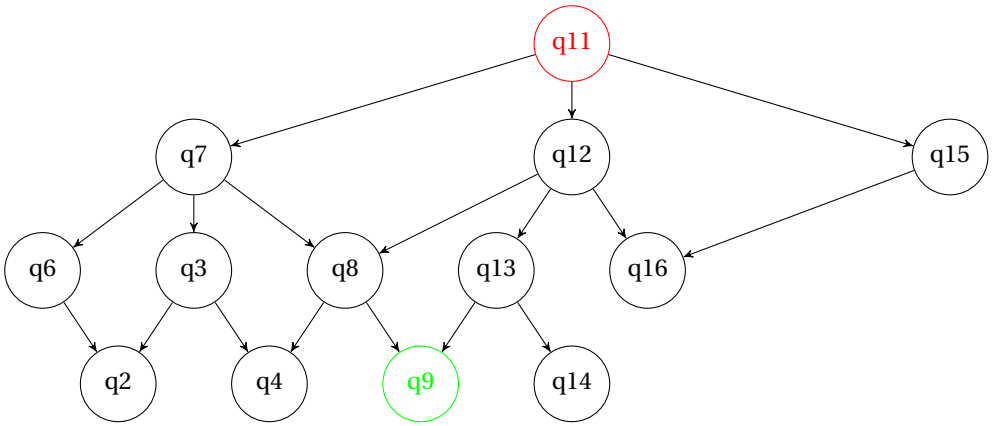


Figure 4.16: Breadth first search (BFS) algorithm illustration based on the steps of Figure 4.12 through 4.15. The BFS algorithm will continue till all the possible paths on the same level have been evaluated.

Different transportation methods

After defining all possible paths, the various transportation methods are introduced. In this thesis, only minimal square lattices with a maximum of 36 qubits and 7, 17 and 49 SC qubit lattices are evaluated. Therefore, as explained in Section 3.4.3, only SWAP based transportation is used. However, due to the different in- and out-latency of the two qubits in a SWAP gate, it is worth to look at moving the control to target qubit and the target to control qubit. The qubit positions are interchanged to evaluate both options. Note that moving a state from A to B, does not differ from moving a state from B to A, but this change affects the occupancy of the qubits (the time at which the qubit is busy). Due to these two alternatives, the inserted SWAP gates may interleave better with earlier gates.

For each path and transportation method, the total latency and the in- and out-latency of each qubit is stored in separate lists. The total latency is defined as the latency of the maximum length of SWAP gates executed in series.

Look-back at earlier instructions

Next, all possible paths need to be evaluated taking the earlier gates of the quantum circuit into account. Inside a quantum circuit, qubits may not be occupied all the time. Therefore there are moments when the qubits are not performing an operation; they are idle. When a path is inserted, it could potentially utilize the idle qubits in earlier cycles to start the transportation earlier. Depending on the paths, this utilization of unused qubits or interleaving with other instructions may potentially be beneficial concerning latency.

To see how much the paths can be interleaved with the previous gates, first buffer2 needs to be updated. This updating is done by reading buffer1 and blocking the 'busy' qubits in buffer2. With the use of buffer2,

each path is checked if and by what extent it can be interleaved with the previous gates to utilize the idle qubits. By what extent each path can be interleaved depends on the other gates. By looking-up in buffer2 if qubits used in the SWAP operation are 'free' before the under consideration two-qubit gate gives a path the possibility to be executed in parallel with previous gates. The total number of cycles that the path can be interleaved with the previous gates is subtracted from the total latency which was the result as discussed in section *different transportation methods*.

Look-ahead at later instructions

The look-ahead part will check whether and to what extent the new mapping of qubits when applying each path affects the future two-qubit gates. Each path will map the qubits differently and will move qubits needed for other two-qubit gates closer or further apart from each other.

For each possible path, the new mapping of qubits is updated to the rest of qubit instructions inside the buffer1. Next, each instruction is checked to assess whether all the two-qubit gates are NN. If this is not the case the minimal distance between the two qubits is stored. The distance is the minimal path length divided by two where both qubits meet in the middle. This distance is then converted to a minimal worst-case latency, the total latency of a SWAP operation times the distance. Finally, the sum of minimal extra latency due to the non-NN two-qubit gates is added to the total latency which was the results as discussed in section *the look-back at earlier instructions* for each path option.

Insert path

After all the paths have been found, different transportation methods have been added, interleaving with previous gates has been assessed, and the effect of possible reordering of future gates has been investigated, the shortest solution with respect to latency can be inserted. This inserting of the shortest solution is done by expanding and updating buffer1 by inserting extra instruction lines containing the SWAP gates.

4.2.3. Rescheduling

The instructions inside the window are being rescheduled with or without insertion of SWAP operations. To do so, buffer2 is updated and used to reschedule the instructions from the beginning of the buffer, till the originally non-NN instruction, using the ASAP strategy. This rescheduling is also performed if all the gates obey the NN constraint. When all gates obey the NN constraint, the rescheduling is performed till where the gates are checked for the NN constraint. The rescheduling is only shifting the operations to a possibly earlier cycle inside the buffer. The rescheduling is done by evaluating if qubits are idle in earlier cycles. By doing so, the rescheduling tries to keep the largest possible slack, largest number of idle qubits, for any possible non-NN gates in the future. The gates are rescheduled ASAP inside the buffer to keep the slack, the possible movement of gates, at the end of the buffer where new instructions are loaded.

Finally, the first instruction line can be written to the QASM output file, and the buffer can be filled up at the end of the buffer to the length of the window size. This process is repeated until the entire QASM code has been routed.

4.2.4. Verification

To see if the routing algorithm is correct two types of verification are performed. First, the circuits with and without routing are executed on the QX-simulator [15]. This simulator executes the quantum circuit on a classical computer. The results of the non-routed quantum circuit should match the results of the routed quantum circuit. Unfortunately, the current version of the QX-simulator does not support any gate-latency. Therefore the exact scheduling of both quantum circuits cannot be checked.

Another method is by removing the inserted paths from the routed circuit. By doing so, the original quantum circuit should be produced as a result. This so-called de-routed quantum circuit has been rescheduled by the routing algorithm. Therefore to evaluate the routing algorithm, the resulting de-routed circuit should

either be rescheduled by a compiler or the evaluation should take this rescheduling into account when comparing it to the original quantum circuit.

4.3. From physical to planar-based logical qubit routing

When going from physical to logical qubit level, the routing algorithm can be adjusted to solve this routing problem. When performing a CNOT gate with planar-based logical qubits, a logical ancilla qubit is needed (see Section 3.4.1). The CNOT operation is always performed at a 90° angle.

To deal with this constraint, the routing algorithm is adjusted by defining the new neighbors based on Figure 4.18. The small dots are the logical ancilla qubits while the large dots are the logical data qubits. When a two-qubit gate between two logical data qubits is performed, and always the same logical ancilla qubit is used, this occupied ancilla qubit does not have to be taken into account for a routing algorithm. Between each pair of two logical data qubits always the same logical ancilla qubit is used. When a two-qubit gate is performed in the same direction as going from the top left corner to its only neighbor it will choose the top right logical ancilla qubit. In the other direction, for example in the bottom left corner, it will use the bottom right qubit. When doing so, a logical ancilla qubit is only used for a dedicated pair of logical qubits. The ancilla is used in different pairs of logical data qubits but is never used at the same time. Because in that situation the logical data qubit is already used for another operation.

Next, the routing algorithm can just be executed in the same as for physical qubits.

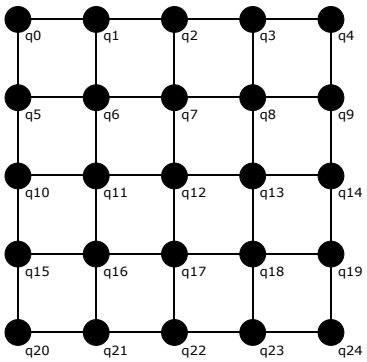


Figure 4.17: Lattice for routing of physical qubits.

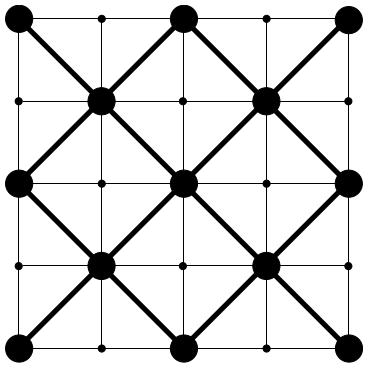


Figure 4.18: Lattice for routing of logical qubits.

4.4. Metrics and quantum circuit characterization

In order to evaluate the performance of the proposed routing algorithm and its different functions, some metrics are needed. After the routing process, extra gates will be added to comply with the NN constraint. These extra gates will most likely add extra latency to the circuit. As previously mentioned, the routing algorithm should try to minimize the added latency, as the larger the execution time, the higher the failure rate of the computation. In addition, it is also important to characterize the quantum algorithm before analyzing the impact of the routing of quantum states. Probably, the lower the percentage of two-qubit gates, the lower the number of routing operations. Although, the actual number of routing operations will depend on the quantum circuit.

The following metrics and characteristics of the quantum algorithms will be used to evaluate the proposed routing algorithm:

- Execution time of the circuit (latency) before and after routing. The difference in execution time is called communication overhead.
- Number of routing events, i.e., the number of two-qubit gates that triggered the routing algorithm.
- Number of SWAPs inserted.
- Qubits temperature map (heat-map), a qubit has a higher temperature when it is used more often.
- Percentage of two-qubit gates in the quantum circuit.

Latency

To derive the execution time before and after routing of a quantum circuit, the duration of each gate needs to be known. In this thesis, some assumptions are made concerning the duration of gates. The gate durations are those as applicable to the superconducting qubits [31]. With these durations of each gate type, the quantum circuit is scheduled and routed.

In Table 4.1 the duration of all the gates are specified for the gates used in physical qubit routing. In Table 4.2 the latencies of the qubits of the various gates can also be found for the physical qubit routing. With those durations, the so-called overhead can be derived for a quantum circuit for a routing algorithm. The durations are translated to cycles by dividing the different execution time by the lowest execution time, 20ns. Therefore in the rest of this thesis 1 cycle corresponds to 20ns execution time.

For routing planar-based logical qubits, the number of ESMs for each gate is needed. In the duration of one ESM, all needed operations are included. Therefore only the number of ESMs are needed for a routing algorithm. In Table 4.3 the number of ESM are specified for each gate performed on a logical qubit. Because the lowest execution time in ESMs is 1, the ESM is directly mapped to cycles used for the routing algorithm.

Table 4.1: Total duration for the various quantum gates for physical superconducting qubits [31].

Gate	**Time [ns]**
X, Y	20
Z, H	40
S, $S^\dagger$, T, $T^\dagger$	60
CNOT	80
SWAP	200
PrepZ	300
Measurement	500

Table 4.2: Latencies for the various qubits of the gates assuming superconducting qubit, 1 cycle has a duration of 20ns [31].

Gate	**IN-latency, q0 [#cycles]**	**OUT-latency, q0 [#cycles]**	**IN-latency, q1 [#cycles]**	**OUT-latency, q1 [#cycles]**
X, Y	0	1	-	-
Z, H	0	2	-	-
S, $S^\dagger$, T, $T^\dagger$	0	3	-	-
CNOT	1	3	0	4
SWAP	1	9	0	10
PrepZ	0	15	-	-
Measurement	0	25	-	-

Table 4.3: Duration in ESMs, of the various quantum gates for planar-based logical qubits assuming a code distance of 3 as in SC-17 [19].

Gate	**ESMs [#]**
X, Y, Z	3
H	12
S, $S^\dagger$	42
T, $T^\dagger$	51
CNOT	9
SWAP	27
PrepZ	3
Measurement	3

Number of routing events

The number of routing events will indicate how many times routing is needed. That is, this number will tell if and how many two-qubit gates with non-NN qubits the routing algorithm found for which a path needs to be found and inserted.

Number of SWAPs

The number of SWAPs can give insight into the quality of the routing algorithm. The number of SWAPs will tell how many transportations are needed between non-neighboring qubits. Most likely will be, the higher the number of routing events, the higher the number of SWAPs to be inserted. But not always the highest number of SWAPs leads to the highest latency as will be explained in the heat-maps.

Heat-map

The heat-map of the qubits will show whether there are certain 'hot-spots', where a lot of qubits need to be transported. This heat-map will tell whether larger paths may be beneficial by routing around these 'hot-spots' and inserting more SWAPs. The heat-map is normalized. This normalizing means that if a physical qubit has a value of '1' it is used in the highest number of all types of gates. When a physical qubit has a value of '0', no gate is performed on this qubit.

Percentage of two-qubit gates

An important characteristic of the quantum circuits for the routing algorithm is the percentage of two-qubit gates. The lower this percentage, the lower the probability of needed transportation.

5

Experiments and results

In this chapter, the designed routing algorithm is analyzed based on the metrics proposed in Section 4.4. The different functions of the routing algorithm are incrementally evaluated. When a comparison is made between different functions, the lowest alternative in a function is selected independently from the window size. This evaluation is done by using proposed random generated benchmarks as well as real quantum benchmarks from ScaffCC and QLib.

In the near feature, different qubit technologies are emerging. Currently, the transmon superconducting qubits has been the most evolving quantum technology [31]. In the coming year, different quantum chips, with 7, 17 and 49 qubits, will be developed. The qubits are arranged in the SC shapes as shown for the SC-7 and SC-17 in figure 3.5. Therefore the different benchmarks are random mapped and routed using a minimal SC qubit lattice size as an input constraint and use the different gate times based on the transmon qubits.

5.1. Random generated benchmarks

When introducing random generated benchmarks, some variables are fixed. Important variables like the distribution of gates, percentage of two-qubit gates, and the locality of two qubits inside the two-qubit gates, influence the routing algorithm. Due to these fixed parameters, possible trends can be discovered. When switching to real quantum benchmarks, the results from these random benchmarks may not always hold. Nevertheless, random quantum benchmarks may give an insight of the overall performance of the routing algorithm. In these random generated benchmarks, the percentage of two-qubit gates is varied. The gates are evenly distributed, and there is no locality as the qubits involved in the two-qubit gates are randomly selected.

The random generated benchmarks have a varying percentage of CNOT gates. For the routing algorithm, only the gate execution times are relevant and not the actual operation they perform on a qubit.The single-qubit gates present in each benchmark are split evenly between X-, Z- and S-gates, which have a duration of respectively one, two and, three cycles as can be seen in Table 4.2. No other types of qubits are used, as they have the same latency. The percentage of gates in the benchmarks can be found in Table 5.1. All random quantum benchmarks contain 7, 17 or 49 qubits. These benchmarks are generated with the use of QPDO built by Ir. L. Riesebos and scheduled as late as possible (ALAP) with the use of the compiler built by Ir. J. van Someren.

Table 5.1: Percentage gates in the random benchmarks with a certain percentage two-qubit gates.

Gate	**Percentage of two-qubit gates [%]**										
	0	10	20	30	40	50	60	70	80	90	100
X	33.3	30.0	26.7	23.3	20.0	16.7	13.3	10.0	6.7	3.3	0.0
Z	33.3	30.0	26.7	23.3	20.0	16.7	13.3	10.0	6.7	3.3	0.0
S	33.3	30.0	26.7	23.3	20.0	16.7	13.3	10.0	6.7	3.3	0.0
CNOT	0.0	10.0	20.0	30.0	40.0	50.0	60.0	70.0	80.0	90.0	100.0

To make sure that all possible gates between two qubits inside the lattice are performed, the number of gates in each benchmark is set to 10000 gates. The expected number of identical two-qubit gates, with the same pair of qubits, in a random benchmark of 10000 gates is given by the formula 5.1, in which n is the number of qubits and $r = 2$ is the sample size, the number of qubits that form a pair. In Table 5.2 the expected number of identical two-qubit gates are specified based on equation 5.1. With all the benchmarks at least every possible control and target qubit combination will be expected to appear in the quantum circuits.

$$C(n,r) = \frac{n!}{r! * (n-r)!} \tag{5.1}$$

Table 5.2: Expected number of identical two-qubit gates in the random benchmarks of 10000 gates.

Qubits [#]	**Percentage of two-qubit gates [%]**									
	10	20	30	40	50	60	70	80	90	100
7	48	95	143	190	238	286	333	381	429	476
17	7	15	22	29	37	44	51	59	66	74
49	1	2	3	3	4	5	6	7	8	9

In the following subsections, first, the effects of the different routing algorithm functions for different windows sizes are analyzed. Next, the effect of possible extra path length is shown. To see the effects of the single-qubit gate duration, different single-qubit gate distributions are also evaluated. Finally, a small extra function is added to give a brief analysis of the effect for a possible different window for the look-ahead function.

5.1.1. Impact of different routing algorithm functions and window sizes

In this section, the routing algorithm performance is evaluated by using the random generated benchmarks. The effect of the different functions of the routing algorithm is investigated by incrementally adding new functions. The following functions are evaluated in the following order:

1. Basic routing ↦ [Basic routing]
2. + Move control and target qubit, meet in the middle ↦ [+ Move C & T]
3. + Split parallel instructions ↦ [+ Split parallel ins.]
4. + Use different transportation techniques ↦ [+ Different tech.]
5. + Look-back to see if the SWAP gates can be interleaved with the previous operations. ↦ [Look-back]
6. + Look-ahead to see how the reordering of qubits will affect future two-qubit gates. ↦ [+ Look-ahead]

The different functions correspond to the different subsections in Section 4.2.2. Function 1, basic routing, inserts the first shortest path by only moving one of the qubits towards the other, e.g., control to target qubit. Function 2 will add the feature of allowing both qubit movement by splitting the paths to meet in the middle. Function 1 and 2 are explained in subsection *Finding paths*. Function 3, split parallel gates, will move parallel single-qubit and two-qubit operations in which qubits are already NN to an earlier cycle as explained in subsection *Split parallel instructions*. Function 4, different transportation techniques, inserts the two standard SWAP alternatives and is explained in subsection *Different transportation methods*. The final two functions, 5 and 6, look-back, and look-ahead take other instructions inside the buffer into account by look-back and look-ahead. These functions are explained in subsections *Look-back at earlier instructions* and *Look-ahead at later instructions* respectively.

For all these functions, different window sizes are tested for the 7 and 17 qubits random benchmarks. Due to the results of these two benchmarks, the 49 qubits random benchmark is only tested with function look-ahead switched on/off and compared to basic routing plus moving control and target qubit. The results are evaluated by the difference in execution time of the quantum circuits with and without routing overhead (added latency) in number of cycles. Also, the number of added SWAPs are shown for each benchmark.

7 qubits random benchmark

The benchmark is evaluated by how the different functions of the proposed routing algorithm impact the total execution time of the quantum circuit (in number of cycles) when considering different window sizes. These results can be found in Figures 5.1 and 5.2 and in Appendix A.1 for random benchmarks with a different percentage two-qubit gates, between 0% and 100%. The green bars represent the lowest total execution time. As expected, the higher the percentage of two-qubit gates, the higher the circuit latency.

As shown in Figures 5.1 and 5.2, the circuit latency decreases as more functionality is added to the routing algorithm. When evaluating the different functions, it is clear that the ability to move both the control and target gives an advantage with respect to only moving the control, as it gives more and shorter path options for evaluation. In Figure 5.1 this function reducing the overhead from 208% (basic routing) to 161% (move target and control), which is an reduction of 15.3% in total execution time. In Figure 5.2 this function reducing the overhead from 364% (basic routing) to 289% (move target and control), which is an reduction of 16.0% in total execution time. The split parallel instructions will only give a minor advantage for high-percentage two-qubit gates and a small disadvantage for low-percentage two-qubit gates when reviewing all the figures in Appendix A.1. This effect can be explained by the fact that the first shortest option is selected. This shortest option may decrease the possible interleaving of future to be inserted paths. The different transportation techniques have a negligible advantage because it will only add the standard SWAP operation with the control and target qubit in both positions as explained in Subsection 4.2.2.

One can also observe that the window size has a varying effect on the circuit execution time. As the percentage of two-qubit gates varies, another window size may be beneficial. For instance, the lowest latency in Figure 5.1 is at windows size of 50 cycles (green bar), whereas, in Figure 5.2 this corresponds to a windows size of 30 cycles. The effects of the different window size can be clarified due to the rescheduling and the two functions, look-back and look-ahead. When the window size is larger, the scheduler can take more gates into account when scheduling the buffer ASAP. The rescheduling result is more optimal and therefore the slack, where new movement instructions can be inserted, is higher. The Look-back functions uses the unoccupied qubits to allow the transportation to start earlier (interleaving with earlier instructions). If the window size is larger, the level of interleaving may increase to a certain level. That is, the level of interleaving depends on the occupancy of the qubits due to earlier operations.

When look-ahead at future gates is enabled, if the window size is larger or the percentage two-qubit gates increases, the positive effect of looking ahead gets smaller. This effect can be explained by the principle that a current optimal path selection is based on the minimal distance between non-NN qubits that need to interact in the future. However, this minimal distance may change when introducing new paths for future two-qubit gate operations as those qubits will maybe be moved to a different position. In other words, when looking ahead, it cannot be predicted what will be the exact position of the qubits involved in future instructions, and it will depend not only on the current inserted path but also on the paths chosen in the future. Therefore, if more instructions are taken into account when looking ahead, possible wrong path selections are inserted, and the benefit of looking-ahead dilutes. Note that, that in this case the optimal size of the window also depends on the percentage of two-qubit gates. If the percentage two-qubit gates is lower, there are fewer two-qubit gates in the same window size, and then a larger window will be more beneficial (see the green bars in Figures 5.1 and 5.2). The green bars have an overhead of 109% and 203% respectively. Compared to the move control and target function, the best results, indicated with the green bars, have a reduction of 19.9% and 22.1% respectively.

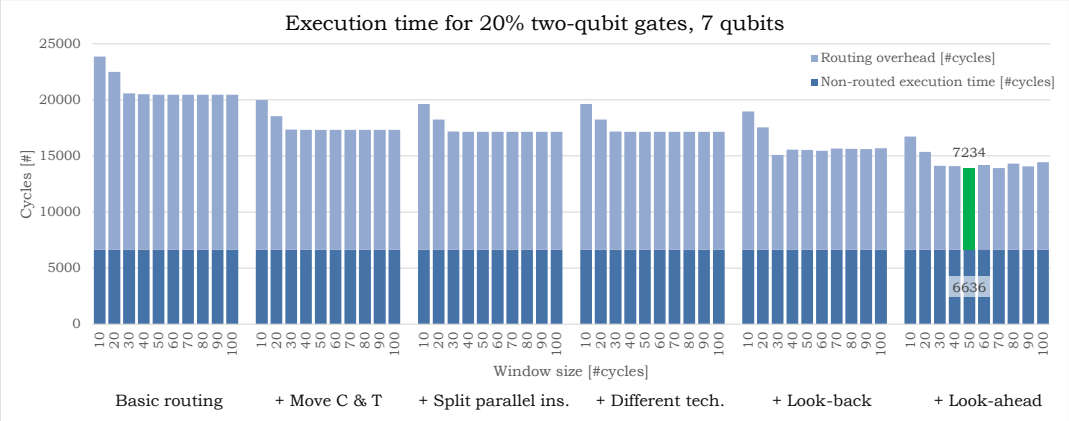


Figure 5.1: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 20% two-qubit gates for 7 qubits.

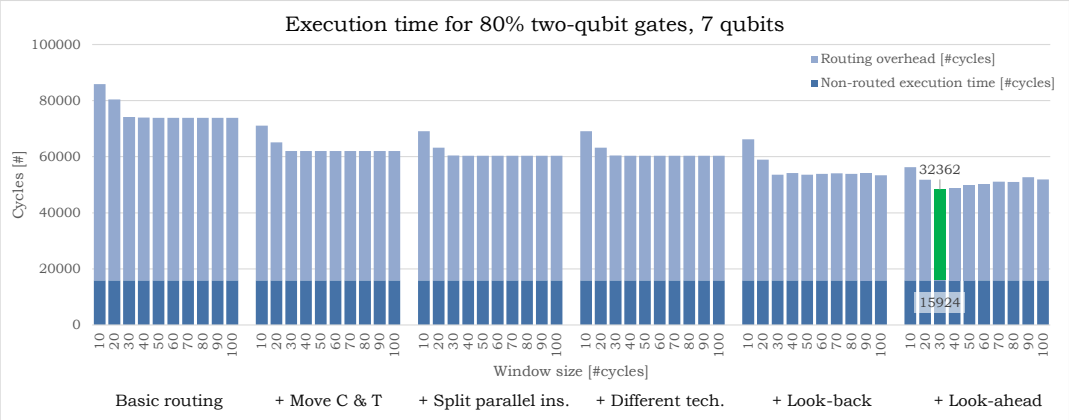


Figure 5.2: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 80% two-qubit gates for 7 qubits.

In Figures 5.3 and 5.4 and in Appendix A.1, the number of routing events and added number of SWAPs for the same 7 qubits random benchmarks can be found. The green bars represent lowest the number of added swaps or number of routing events. For the first four functions, only one bar is plotted as different window size does not affect the selection of a path. Note that these functions, do not take other gates into account when selecting the shortest path. When looking at the different functions, all the functions (except for the look-ahead function) have a limited effect on the number of added SWAPs and routing events. The split parallel instructions function has a small benefit because instructions with NN qubits is executed first. It reduces the number of added swaps by around 3% (compared basic routing with split parallel instructions). Therefore possible extra routing events and number of SWAPs are avoided. The look-ahead function has a major impact on the total number of added SWAPs, around 30% reduction (compared split parallel instructions with look-ahead). This result can be explained by the fact that only the look-ahead function will look at multiple two-qubit gates and tries to optimize the overall qubit movement. The other functions only try to optimize the total latency by interleaving or executing NN-gates in parallel earlier.

When evaluating the different window sizes for the different functions, as mentioned, the first four functions will always insert the same path due to the principle that no other instructions are taken into account. The different window sizes have a minor influence on the look-back function. The look-ahead function tries to optimize the overall qubit movement. As previously mentioned at the results on the execution time, when more two-qubit gates are taken into account for the path decision, possible wrong paths are selected. Therefore there is an optimal number of two-qubit gates inside the buffer to give the lowest number of added SWAPs.

From these graphs, it can be seen that the minimal number of routing events and SWAPs does not always relate to the shortest latency. When optimizing for the number of SWAPs the shortest paths are selected which may be in favor of total latency. This optimization is better visible for a larger percentage of two-qubit gates. When increasing the percentage of two-qubit gates, a smaller window size results in the lower number of SWAPs being added. This effect occurs because if more two-qubit gates are checked for their minimal distance, the effect is balanced out just when evaluating the overall added latency. If more minimal distances between two-qubit gates are added to the latency, the overall movement will be constant.

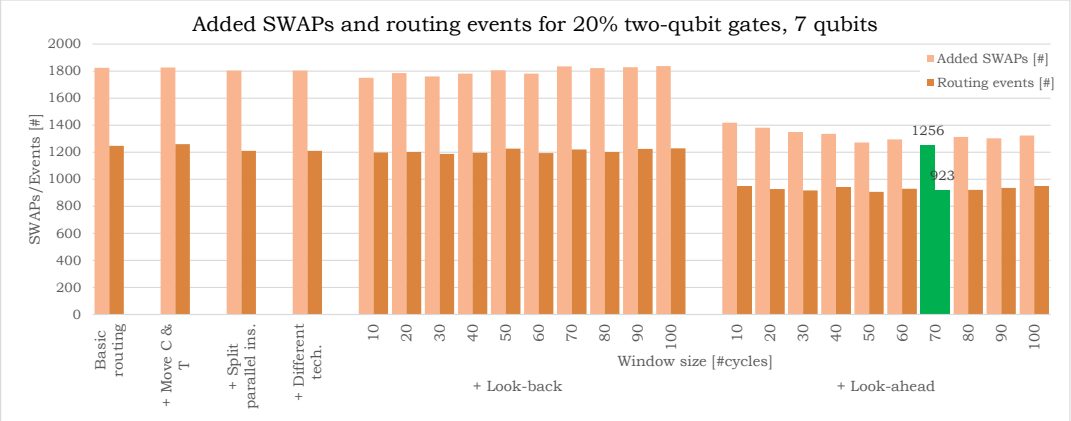


Figure 5.3: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 20% two-qubit gates for 7 qubits.

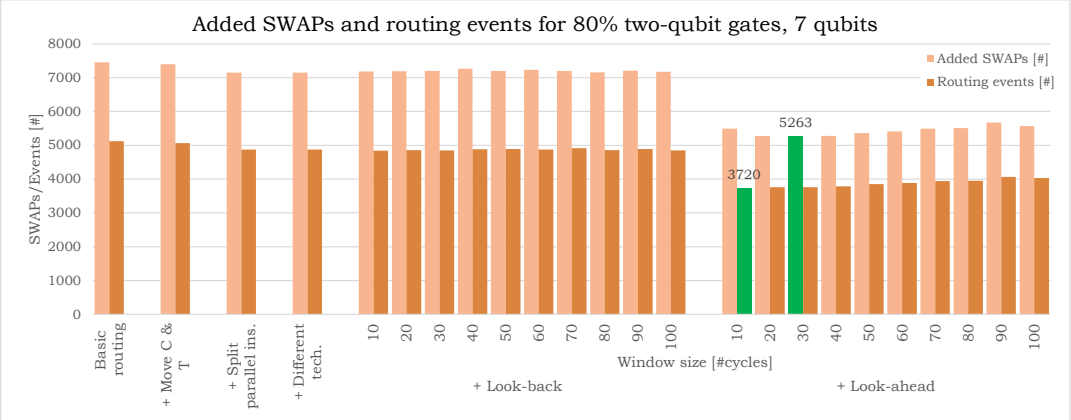


Figure 5.4: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 80% two-qubit gates for 7 qubits.

17 qubits random benchmark

To see the effects of different window sizes for a 17 qubits random benchmark, each added function is analyzed with a varying window size. These results can be found in Figures 5.5 and 5.6 and in Appendix A.2 for random benchmarks with a different percentage two-qubit gates, between 0% and 100%. The green bars represent the lowest total execution time that corresponds to an overhead of 284% and 472% respectively. This overhead is a large reduction compared to moving target and control where 391% and 648% overhead respectively is introduced. The total execution time is reduced by 21.8% and 23.6% respectively.

An initial observation from these figures is that adding functions to the routing algorithm produces similar overall latency trends in the 17 qubits version compared to the 7 qubits version. A larger window size will improve the rescheduling of the quantum instructions because more gates are taken into account. This improved rescheduling is now even better visible due to the 'exponential' decay in execution time for the first

four functions. A smaller window size will be more beneficial with respect to the look-ahead function due to the limited number of future two-qubit gates that are taken into account as can be seen in Appendix A.2.

The major difference between the 7 and the 17 qubits benchmarks can be seen for higher percentage two-qubit gates. From 50% two-qubit gates onwards, as can be seen in Figure 5.6, the routing algorithm is more effective, concerning latency, when the look-ahead function is switched off. This effect is a result of the same phenomenon which causes a smaller window size to be more effective for the look-ahead function. When look-ahead is enabled, and the percentage of two-qubit gates is increased, the advantage of look-ahead is diminished with respect to latency.

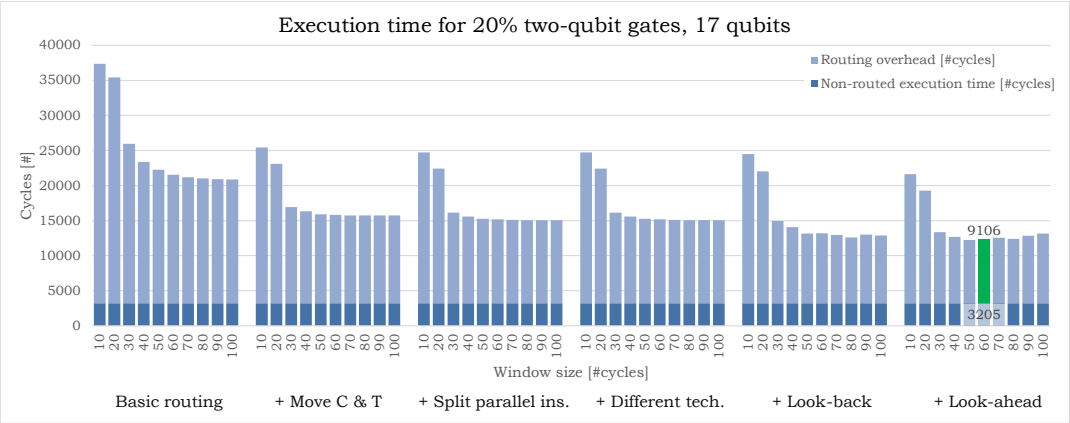


Figure 5.5: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 20% two-qubit gates for 17 qubits.

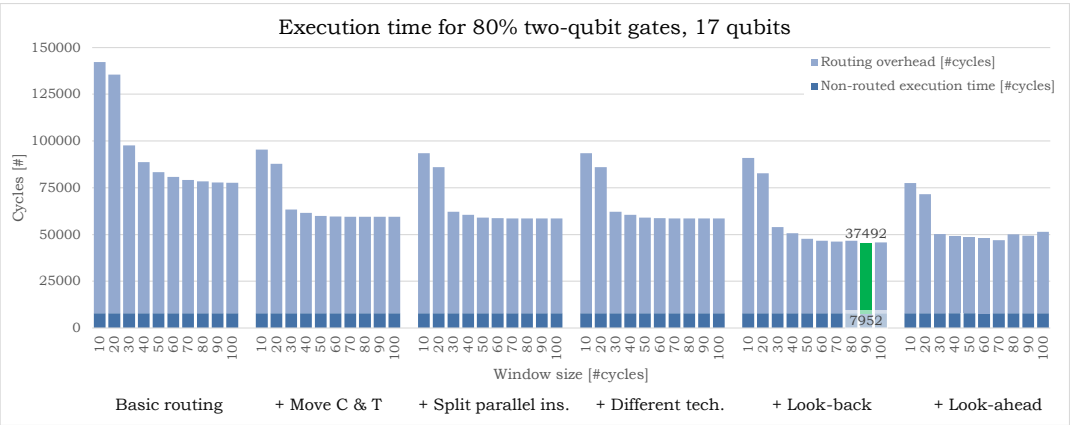


Figure 5.6: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 80% two-qubit gates for 17 qubits.

In Figures 5.7 and 5.8 and in Appendix A.2, the number of routing events and added SWAPs for the same 17 qubits random benchmarks can be found. The same effects on the 7 qubits random benchmark hold for the 17 qubits random benchmark. The green bars represent the lowest number of added swaps or number of routing events. The number of added SWAPs is reduced with around 32% compared to basic routing.

In these graphs can be seen that the minimal number of routing events and SWAPs does not always map the shortest latency. The look-ahead function is optimizing the total number of SWAPs. Therefore the shortest paths are selected, and this may be in favor of total latency. In all the cases the look-ahead function will make sure that the total number of SWAPs is at its lowest. This effect becomes even more visible for two-qubit gate percentages above 50%.

When looking at the different window sizes for the look-ahead function with the increasing percentage two-qubit gates, the most optimal solution concerning SWAPs is at a lower window size. This effect is in line with the results from the 7 qubits random benchmarks. When the percentage two-qubit gates is increased, the window size should be lower to have the same proportion two-qubit gates inside the buffer.

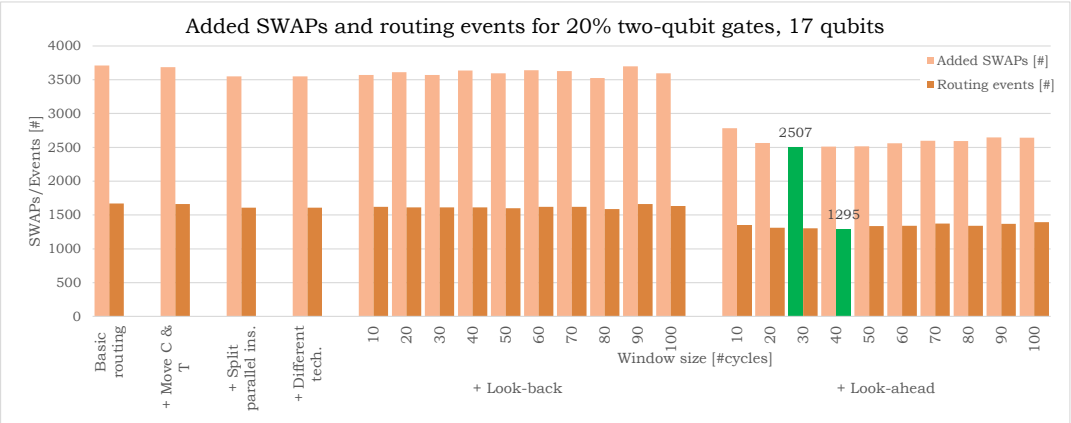


Figure 5.7: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 20% two-qubit gates for 17 qubits.

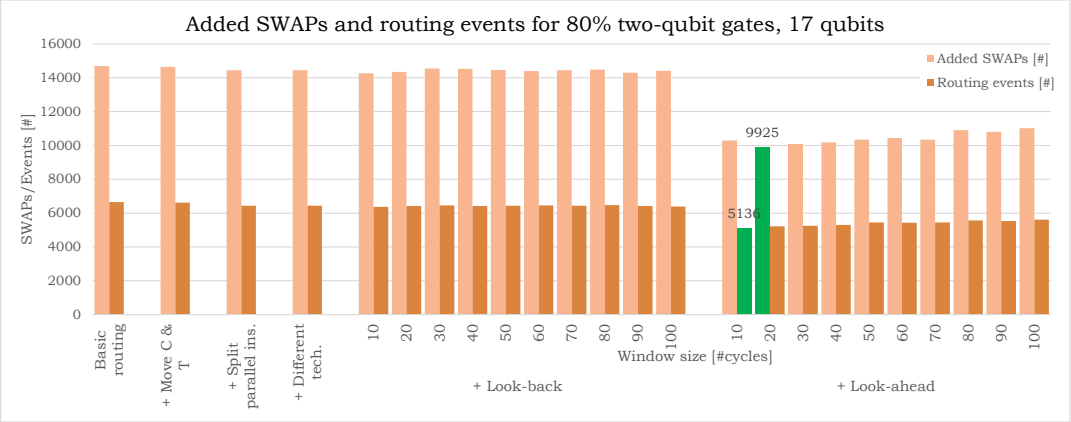


Figure 5.8: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 80% two-qubit gates for 17 qubits.

49 qubits random benchmark

To see the effects of different window sizes for a 49 qubits random benchmark, the last two added functions, look-back, and look-ahead, are analyzed with a varying window size. Due to the results of the 7 and 17 qubits random benchmarks, only the last two functions are enabled and compared to the basic routing including the movement of control and target qubits. These results can be found in Figures 5.9 through 5.12 and in Appendix A.3 for random benchmarks with a different percentage two-qubit gates, between 0% and 100%. The green bars represent the lowest alternative. For the execution time, the green bars correspond to an overhead of 791% and 1154% respectively. Compared to moving target and control where respectively 1045% and 1611% overhead is inserted. The total execution time is reduced by 22.1% and 26.7% respectively.

Just as in the 7 and 17 qubits random benchmarks, the larger window size is beneficial for rescheduling as can be seen in Figures 5.9 and 5.11. The exponential decay in execution time with increasing window sizes is even better visible. Due to the increase in qubits, the routing algorithm is executed with a window size up to 150 cycles. The look-ahead function is more advantageous with lower window sizes. This optimal window size again decreases when the percentage of two-qubit gates is increased. This effect is the same as for the 7 and 17 qubits random benchmarks.

Due to this large number of qubits, the look-ahead function is never beneficial with respect to latency. This effect is even larger compared to the 17 qubits random benchmarks. Due to the larger number of qubits, a lot of moment needs to be added. There is no locality inside the instructions so a lot of transportation is needed and when look-ahead, even less CNOT gate evaluations should be taken into account. The total number of added SWAPs, however, does always benefit from the look-ahead function. The explanation for this observation is the same as for the 7 and 17 qubits random benchmarks although the optimal window sizes are even smaller as can be seen in Figures 5.10 and 5.12

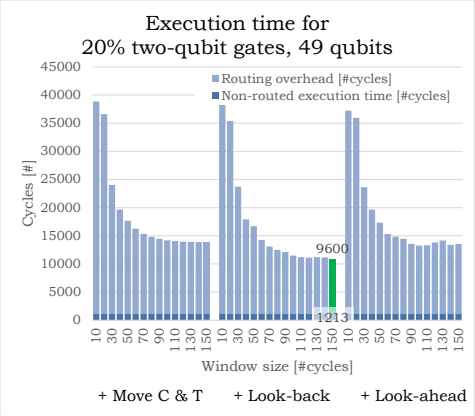


Figure 5.9: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 20% two-qubit gates for 49 qubits.

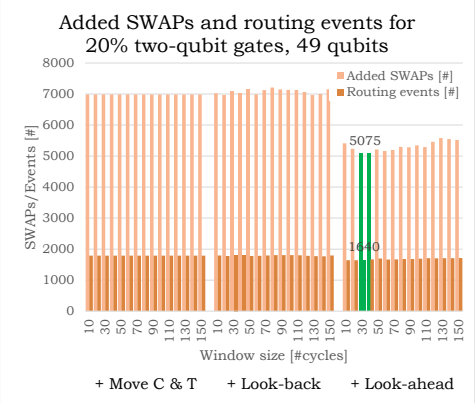


Figure 5.10: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 20% two-qubit gates for 49 qubits.

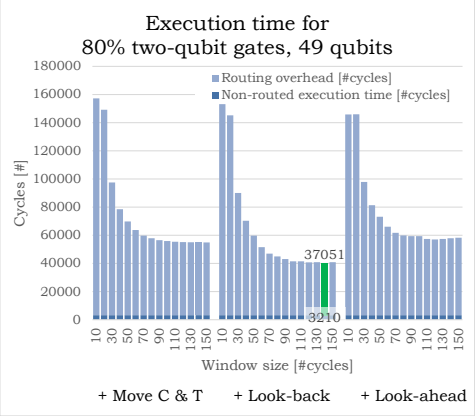


Figure 5.11: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 80% two-qubit gates for 49 qubits.

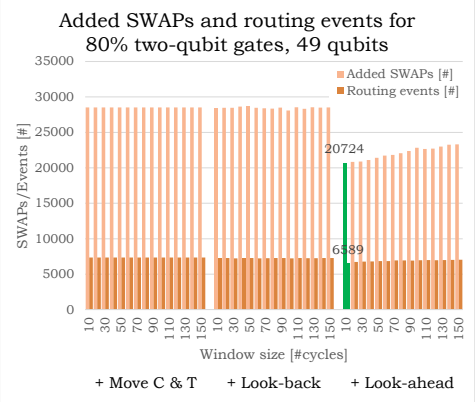


Figure 5.12: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 80% two-qubit gates for 49 qubits.

Overview

After running the random benchmarks with a varying percentage of two-qubit gates and a varying number of qubits more data can be evaluated. With the results from this subsections, the minimal routing overhead in percentage is plotted in Figure 5.13. From this figure can be seen that the overhead has a logarithmic relationship between the percentage overhead and percentage two-qubit gates. Between the different number of qubits, there seems to be a linear relationship based on those three different number of qubit results.

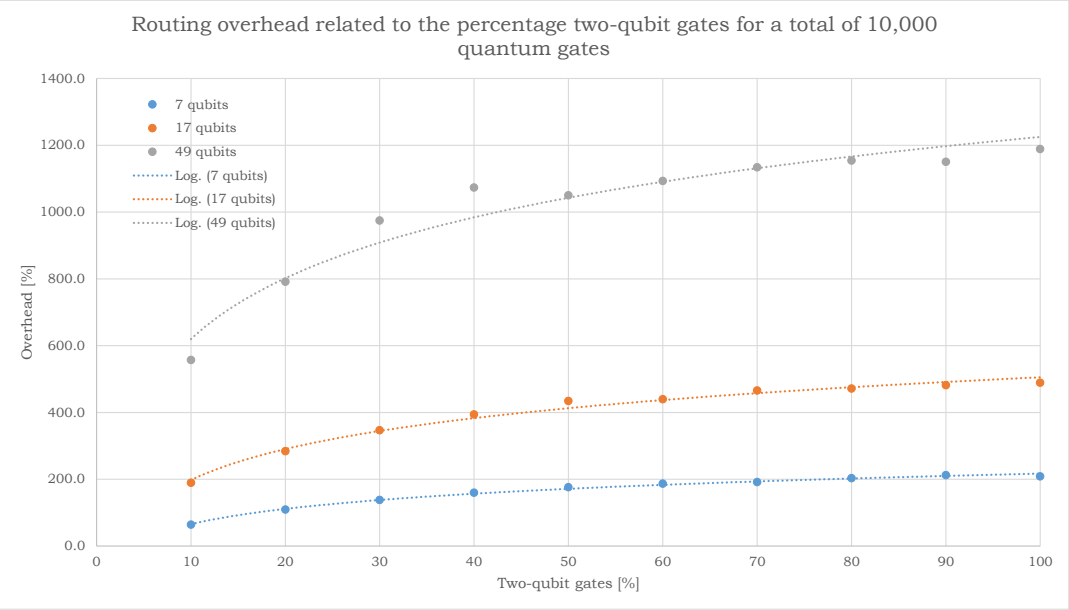


Figure 5.13: Minimal routing overhead in percentage for different number of qubits and percentages of two-qubit gates.

For the minimal number of added SWAPs also the data is plotted from subsections 5.1.1 and 5.1.2. Figure 5.14 shows how the number of added SWAPs is related to the percentage two-qubit gates. It can be seen that the relation between those two variables is linear. Also, the relation between the number of qubits and the number of added SWAPs seems to be linear.

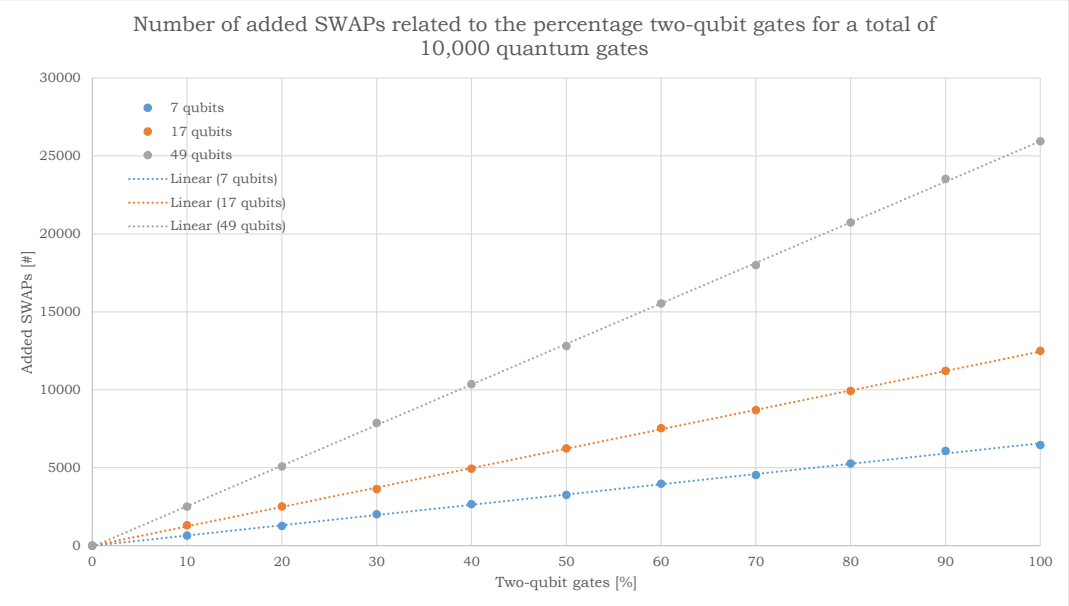


Figure 5.14: Minimal number of added SWAPs for different number of qubits and percentages of two-qubit gates.

Heat-maps

To visualize how much qubits are used because of routing, a few heat-maps are generated. Only three versions are visualized to see the effects of the look-back and look-ahead functions. In Figures 5.15 through 5.17 the heat-maps of the 17 qubits circuits with 20% two-qubit gates are simulated. In these heat-maps can

be seen that the difference between the output circuits is very similar. When enabling only the look-back function, the average qubit occupancy rate spreads out a bit. The look-ahead function counteracts this effect because this function tries to minimize the number of SWAPs. This effect can also be seen with higher percentage of two-qubit gates as can be seen in Figures 5.18 through 5.20. With increasing percentage two-qubit gates, more SWAPs are needed. Therefore the qubits in the middle of the lattice have a higher occupancy rate compared to the qubits on the edge of the lattice.

The effects for the 17 qubits random circuits also hold for the 49 qubits circuits. The heat-maps of the 20% two-qubit gate 49 qubits version can be seen in Figures 5.21 and 5.22. The heat-maps of the 80% two-qubit gate 49 qubits version can be seen in Figures 5.23 and 5.24.

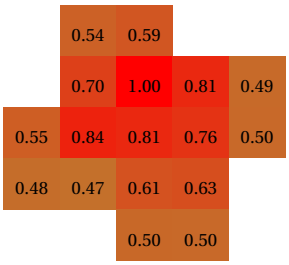


Figure 5.15: heat-map: 17 qubits, 20% two-qubit gates (window size 60) without look-back and look-ahead

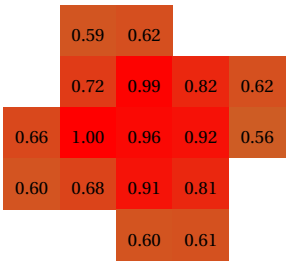


Figure 5.16: heat-map: 17 qubits, 20% two-qubit gates (window size 60) with look-back and without look-ahead

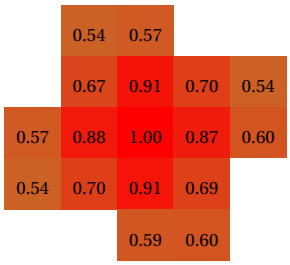


Figure 5.17: heat-map: 17 qubits, 20% two-qubit gates (window size 60) with look-back and look-ahead

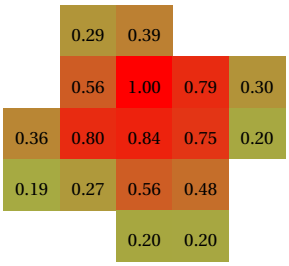


Figure 5.18: heat-map: 17 qubits, 80% two-qubit gates (window size 60) without look-back and look-ahead

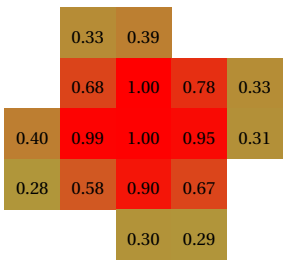


Figure 5.19: heat-map: 17 qubits, 80% two-qubit gates (window size 60) with look-back and without look-ahead

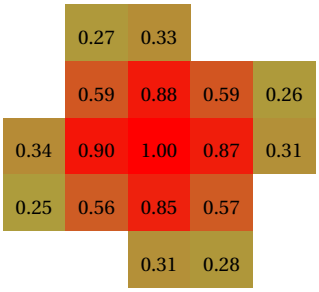


Figure 5.20: heat-map: 17 qubits, 80% two-qubit gates (window size 60) with look-back and look-ahead

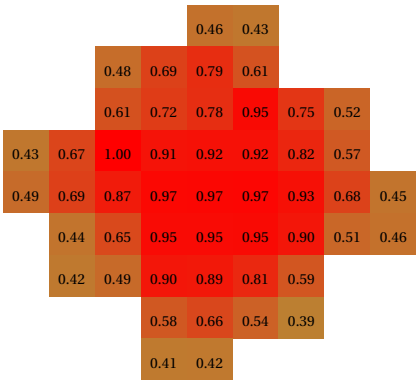


Figure 5.21: heat-map: 49 qubits, 20% two-qubit gates (window size 40) with look-back and without look-ahead

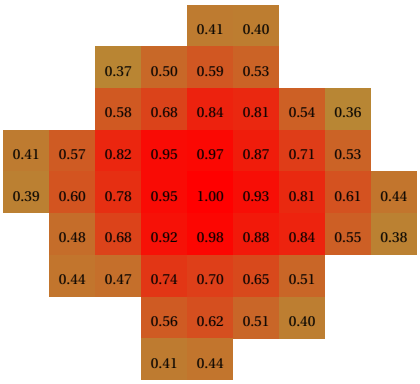


Figure 5.22: heat-map: 49 qubits, 20% two-qubit gates (window size 40) with look-back and look-ahead

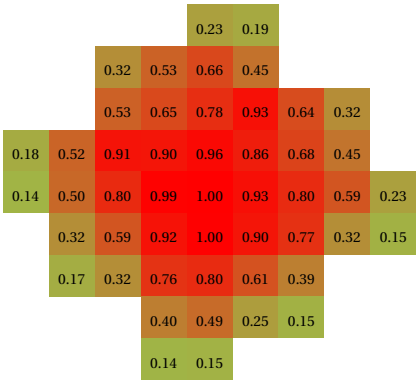


Figure 5.23: heat-map: 49 qubits, 80% two-qubit gates (window size 40) with look-back and without look-ahead

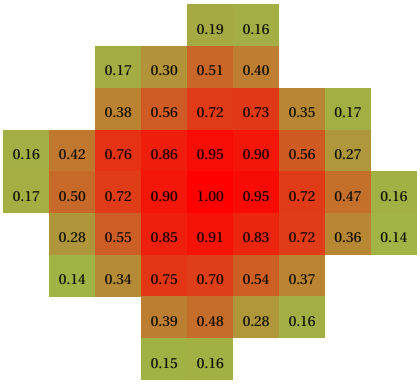


Figure 5.24: heat-map: 49 qubits, 80% two-qubit gates (window size 40) with look-back and look-ahead

Conclusions

In this subsection, the effects of the different functions of the proposed routing algorithm have been analyzed using random generated benchmarks. The random benchmarks contained 7, 17 and 49 qubits and were executed on a 7-, 17- and 49-qubit SC plane lattice respectively where all qubits inside the lattice are physical qubits. The different functions were enabled sequentially. The split target and control function and different transportation techniques function generate more path options which are evaluated in the look-back and look-ahead function where other gates are taken into account. Split parallel instructions will execute NN-gates before the non-NN-gate by shifting the NN-gates and single-qubit gates to an earlier cycle. The look-back function will look at possible interleaving the SWAPs with previous gates. Finally, look-ahead adds to the total latency, depending on a selected path, all the minimal distance between the other two-qubit gates, calculated in a number of SWAPs and therefore cycles.

From the results, it was clear that the split target and control function, in which both the target and control are moved at the same time, has a major advantage. This process gives the routing algorithm more and shorter alternatives. The different transportation techniques function is not effective for these small lattice sizes where only standard SWAP gates are used as explained in Section 4.2.2. The split parallel instructions function is most of the times by itself a bit effective, although the strength of this function is better visible when the look-back and look-ahead functions are enabled. When only the split parallel instructions function is enabled, there are still multiple shortest paths. In that situation, the first shortest solution in the list is selected.

The look-back function is very effective in interleaving the shortest path with previous instructions and will always be advantageous to use. The window size affects this look-back function in the sense that if qubits are not occupied before the instructions inside the buffer, the path could have been interleaved even further. The look-ahead function is based on the principle of evaluating the total minimal distance between the other two-qubit gates. This function minimizes the number of SWAPs and not directly the latency. Therefore the look-ahead function, when increasing the number of qubits and the two-qubit gate percentage, will not be beneficial with respect to latency, although the number of added SWAPs will always be lower when using the look-ahead function.

When looking at the different window sizes, due to rescheduling a larger window size is usually beneficial because more gates are taken into account. But the larger window is less beneficial when looking ahead. When more two-qubit gates are taken into account, more minimal distances between the two-qubit gates are added to the decision. A look-ahead decision is based on all two-qubit gates inside the buffer. When a path is selected, and the next non-NN qubits are routed, the position of qubits changes. Therefore gates far in the future are reordered multiple times. A decision based on these far future two-qubit gates has a negative effect because the routing is balanced out between all qubits. Less optimal paths may then be selected. Therefore a shorter window size is more beneficial when enabling the look-ahead function.

5.1.2. Different path lengths

Apart from the window size, another parameter influencing the routing algorithm performance is a path length adjustment. In Section 5.1.1 the path length was set to the minimal path length in which only minimal path length options were inserted by the routing algorithm. In this section the path length is varied, that is minimal path length plus a constant are allowed. This option allows detours around occupied qubits. These detour options are only useful for the look-back and look-ahead function because only these functions take into account other qubit gates. Therefore only the look-back and look-ahead function are evaluated.

7 qubits random benchmark

In Figures 5.25 and 5.26 and in Appendix B.1, both the execution time as well as the number of routing events and SWAPs are plotted. In these graphs, only the best option related to the window size is stored for a certain path length. It is clear to see that this extra path length is never beneficial with respect to latency or the number of added SWAPs compared to the 'zero' extra path length. The actual increase of SWAPs can be explained by the fact that on a short-term, a detour may be beneficial with respect to SWAPs, but this will lead to a longer latency in the end.

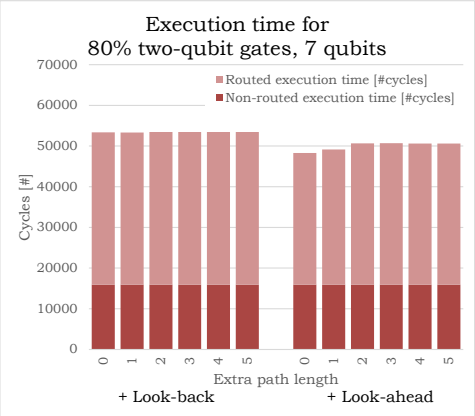


Figure 5.25: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 80% two-qubit gates for 7 qubits.

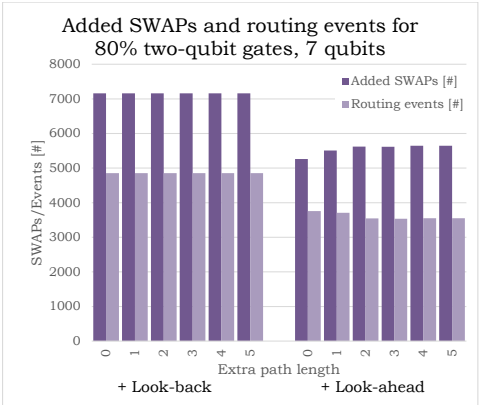


Figure 5.26: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 80% two-qubit gates for 7 qubits.

17 qubits random benchmark

In Figures 5.27 and 5.28 and in Appendix B.2, both the execution time as well as the number of routing events and SWAPs are plotted. In these graphs, only the best option related to the window size is stored for a certain

path length. The effects on the 17 qubits random benchmark are the same as for the 7 qubits random benchmark. In summary, taking detours is not beneficial concerning latency or added SWAPs inside this quantum circuits.

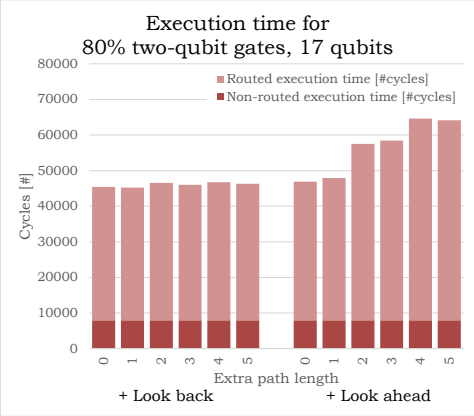


Figure 5.27: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 80% two-qubit gates for 17 qubits.

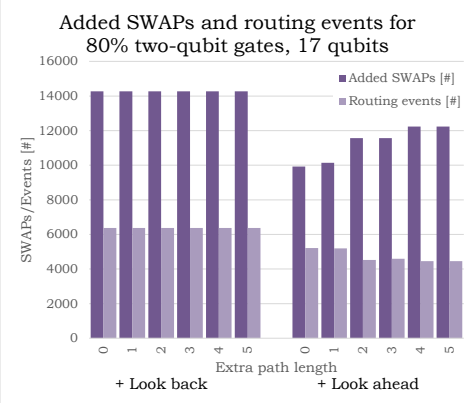


Figure 5.28: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 80% two-qubit gates for 17 qubits.

49 qubits random benchmark

In Figures 5.29 and 5.30 and in Appendix B.3, both the execution time as well as the number of routing events and SWAPs are plotted. In these graphs, only the best option related to the window size is stored for a certain path length. When evaluating the 49 qubits random benchmark, the possibility of taking detours sometimes has a minor advantage. When for example looking at the benchmark in which 80% of gates are two-qubit gates, the extra path length of 1 is a bit faster, 5.6% when comparing zero path length with one extra path length when look-ahead is enabled. Although, the fastest option is with 'zero' extra path length and look-ahead disabled.

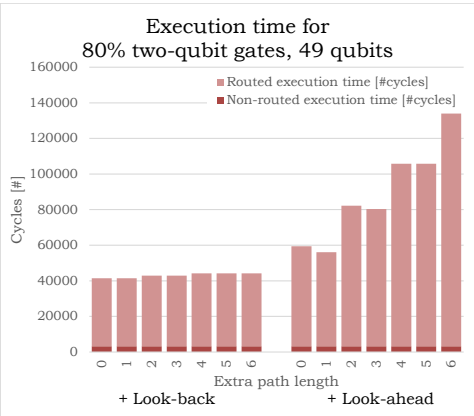


Figure 5.29: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 80% two-qubit gates for 49 qubits.

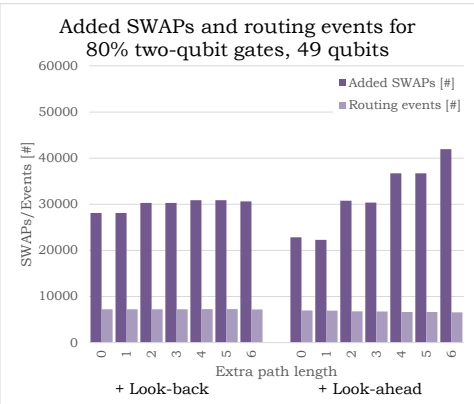


Figure 5.30: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 80% two-qubit gates for 49 qubits.

Conclusions

In this subsection, the effect of allowing additional path length has been analyzed. This extra path length allows detours for occupied qubits. The results show that with the proposed routing algorithm, taking detours is nearly never beneficial. In some situations for the 49 qubits random benchmark, the extra path length had a minor positive effect. The major reason why this effect was limited is that the look-ahead function does not take into account whether qubits are occupied. Only the minimal qubit distance translated to latency is evaluated when selecting the shortest path. Therefore taking detours may be more beneficial when taking into account the occupancy of the qubits.

5.1.3. Different single-qubit gate distributions

In subsections 5.1.1 and 5.1.2 the different latencies of single-qubit gates featured an equal number of appearance inside the benchmarks. Since the average single-qubit gate duration is two cycles, extra random benchmarks are generated with only one type of single-qubit gate, either 'X', 'Z' or 'S'. These gates have a duration of one, two and three cycles respectively. See Tables 4.1 and 4.2 for all gate durations. For each type of gate, the percentage two-qubit gates is varied.

The random benchmarks with a single type of single-qubit gates are only for 17 qubits. In Figures 5.31 through 5.34 and in Appendix C, the results are plotted with respect to number of cycles, the added number of swaps and the number of routing events. From these graphs, it can be seen that for a constant two-qubit gate percentage the number of added SWAPs is roughly the same. The routing overhead in cycles gets lower with increasing single-gate duration, due to the higher possibility to interleave. The conclusions drawn in subsections 5.1.1 and 5.1.2 hold for these benchmarks.

When increasing the single-qubit latency the lowest number of added SWAPs is at a higher window size. This effect can be explained by the fact that the number of two-qubit gates inside the window will be lower in case the single-qubit gate latency increases. This effect is not very consistent for lower percentages of two-qubit gates but clearly visible in Figures 5.31 through 5.34.

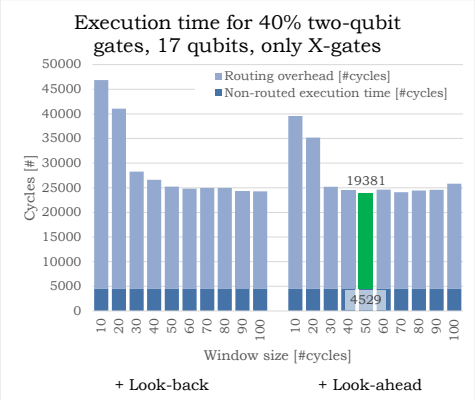


Figure 5.31: Quantum circuit execution time in cycles. The random benchmark contains 60% X-gates and 40% two-qubit gates for 17 qubits.

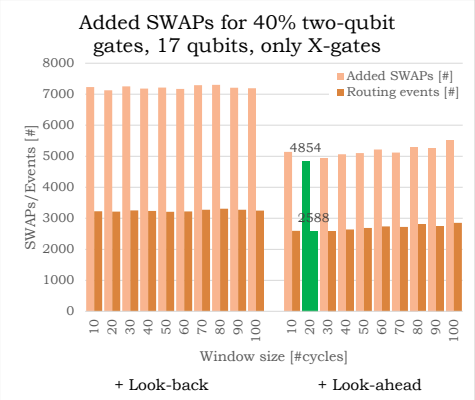


Figure 5.32: Number of routing events and added SWAPs. The random benchmark contains 60% X-gates and 40% two-qubit gates for 17 qubits.

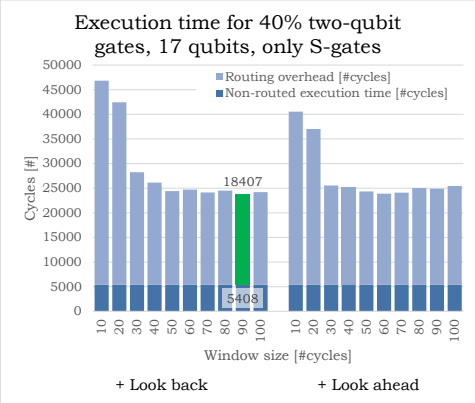


Figure 5.33: Quantum circuit execution time in cycles. The random benchmark contains 60% S-gates and 40% two-qubit gates for 17 qubits.

Figure 5.34: Number of routing events and added SWAPs. The random benchmark contains 60% S-gates and 40% two-qubit gates for 17 qubits.

5.1.4. Variable look-ahead window

The proposed routing algorithm has a fixed window size of 150 cycles for the rescheduling and all the functions inside the routing algorithm. Therefore a small test is performed to see whether the routing overhead will benefit from a variable look-ahead window when keeping the window for rescheduling instructions fixed. This option may benefit from a better look-ahead decision by evaluating a few two-qubit gates and taking more gates into account for rescheduling. This variable look-ahead window defines how far the look-ahead function will look inside the buffer to other two-qubit gates. Only the minimal distances between the other two-qubit gates in the look-ahead window are taken into account.

In Figure 5.35 a plot is shown for the random benchmark with 49 qubits and 50% two-qubit gates. With a look-ahead window size of zero, no look-ahead is used. With increasing window size more instructions are taken into account when looking ahead. From this figure, one can easily see that the look-ahead window with this high number of qubits is never beneficial with respect to latency. This effect can be explained by the fact that the look-ahead function does not take into account occupation of the individual qubits as well as the disorder of future qubits when more paths are inserted. As a result, the routing path is not checked for possible interleaving with future gates. From this one can conclude that the look-ahead function is optimized for minimizing the number of SWAPs, which is not always beneficial for finding the lowest latency.

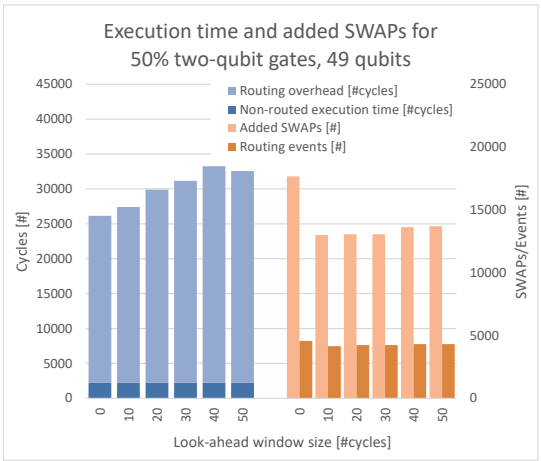


Figure 5.35: Quantum circuit execution time in cycles and number of added SWAPs for different window sizes for look-ahead. The buffer window size is fixed 150 cycles. The quantum circuit consists of 49 qubits and contains 50% two-qubit gates.

5.2. Real quantum benchmarks

To see how the routing algorithm operates under real quantum circuits some ScaffCC and QLib benchmarks are used. These benchmarks are routed on a minimal SC-size qubit lattice. The ScaffCC and QLib benchmarks are evaluated in Sections 5.2.1 and 5.2.2 respectively.

5.2.1. ScaffCC benchmarks

Two real scheduled ScaffCC benchmarks [12] are mapped using the routing algorithm with a random initial placement. Only the Ising Model and Square Root algorithms from the ScaffCC benchmarks are analyzed because only these two benchmarks can currently be compiled by OpenQL. The specifications of the Ising Model and Square Root algorithms can be found in Table 5.3. For both algorithms, no extra path length is used. The two ScaffCC algorithms are mapped with no initial placement on a minimal SC size lattice. Therefore the qubits start from the top left and first mapped in the x-direction and then in the y-direction.

Table 5.3: Specifications ScaffCC benchmarks.

Benchmark	Gates [#]	Two-qubit gates [%]	Qubits [#]	Lattice size
Ising Model	52668	0.2	10	SC-17
Square Root	18370	39.2	30	SC-49

In Figures 5.36 and 5.37 the circuit execution time and added SWAPs results are plotted for the Ising Model algorithm. From these graphs can be seen that the overhead is pretty low because of the low percentage of two-qubit gates. Because the number of inserted SWAPs is extremely low, the added overhead could probably be improved with a better initial placement.

In Figures 5.38 and 5.39 the circuit execution time and added SWAPs results are plotted for the Square Root algorithm. In these graphs can be seen that the same effects return as with the random benchmarks from Section 5.1. The lowest overhead is obtained with a large window size and the look-ahead function disabled. By using the routing algorithm, the overhead is reduced from 161% to 121% compared moving control and target and the shortest option. This reduction relates to a reduced execution time of 15.3%. What is interesting to see is when looking at the heat-maps in Figure 5.40 and 5.41, there is a slight difference between the look-back and look-ahead functions visible. The heat-map of the look-ahead function switched on shows a more evenly distributed load among all the qubits. Some qubits are in not used at all, labeled with a zero because no extra path length is allowed, so these qubits cannot be utilized.

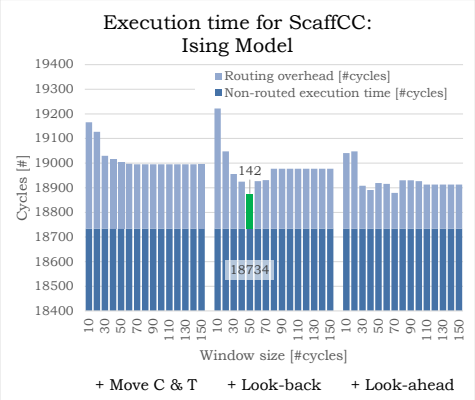


Figure 5.36: Quantum circuit execution time in cycles for different functions and window sizes for the ScaffCC: Ising model algorithm.

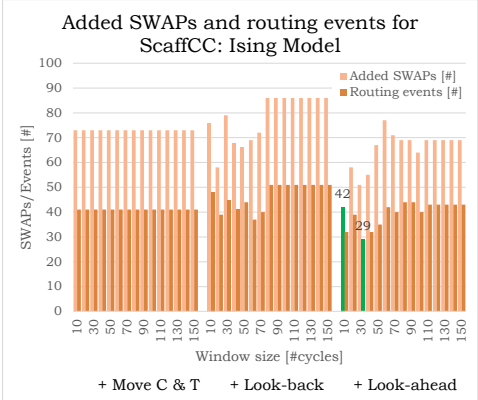


Figure 5.37: Number of routing events and added SWAPs for different functions and window size for the ScaffCC: Ising model algorithm.

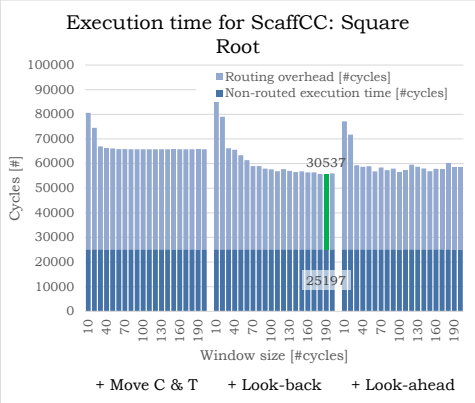


Figure 5.38: Quantum circuit execution time in cycles for different functions and window sizes for the ScaffCC: Square Root algorithm.

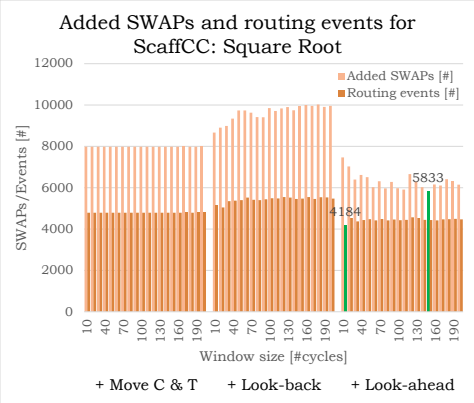


Figure 5.39: Number of routing events and added SWAPs for different functions and window size for the ScaffCC: Square Root algorithm.

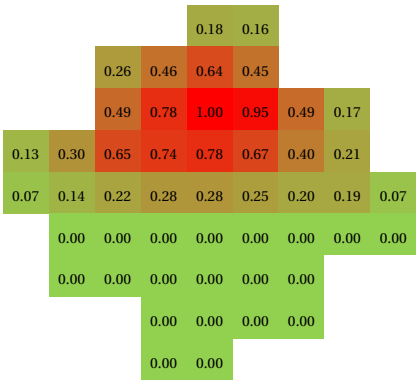


Figure 5.40: heat-map: ScaffCC: Square Root algorithm (window size 60) with look-back and without look-ahead

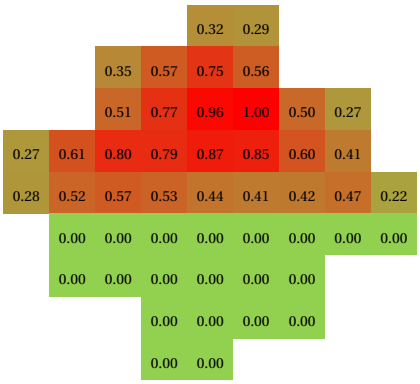


Figure 5.41: heat-map: ScaffCC: Square Root algorithm (window size 60) with look-back and look-ahead

5.2.2. QLib benchmarks

In this section, the routing algorithm is evaluated with some QLib benchmarks [20]. The Bernstein-Vazirani Search (BVSearch), Cuccaro Adder, Multiplier, Quantum Modular Exponentiation (QME) and the Adder from Vedral, Barenco, and Ekert (VBE Adder) are used to test the routing algorithm. Only these quantum algorithms were currently available and can currently be compiled by OpenQL.

The functionality of those benchmarks is not relevant for this thesis. The only relevant complexity is given by some variables. q is the number of qubits the circuit is based on (not to be confused with the total number of used qubits). This number means that a quantum circuit is made to add q number of qubits although the circuit may use more qubits to perform this circuit. N is a to be factored number. a is a constant random number between 2 and $N-1$, which is co-prime to N. These parameters define the number of qubits, percentage two-qubit gates and the locality and distribution of the two-qubit gates inside the benchmarks. The specifications of the different benchmarks can be seen in Table 5.4.

The different QLib algorithms are mapped with no initial placement on a minimal SC size lattice. Therefore the qubits start from the top left and first mapped in the x-direction and then in the y-direction. Also, no extra path length is used. By doing so, the results can be compared to the previous results of the ScaffCC benchmarks in Subsection 5.2.1.

Table 5.4: Specifications QLib benchmarks.

Benchmark		Gates [#]	Two-qubit gates [%]	Qubits [#]	Lattice size
BVSearch	q17 100	41	7.3	18	SC-17 + 1
	q49 10000	106	5.7	49	SC-49
	q49 1893754	115	11.3	50	SC-49 + 1
Cuccaro Adder	q5	181	44.8	12	SC-17
	q7	253	44.7	16	SC-17
	q15	541	44.6	31	SC-49
	q23	829	44.5	48	SC-49
Multiplier	q3	871	40.2	16	SC-17
	q5	2478	40.8	26	SC-49
	q7	4910	41.1	36	SC-49
	q8	6435	41.2	41	SC-49
	q9	8166	41.2	46	SC-49
QME	a4 N15	39822	37.0	19	SC-49
	a5 N6	16529	36.9	15	SC-17
	a5 N21	78117	37.0	23	SC-49
	a7 N15	39822	37.0	19	SC-49
VBE Adder	q5	306	41.2	16	SC-17
	q15	986	41.2	49	SC-49

In Figures 5.42 and 5.43, and the figures in Appendix E, the execution time and added SWAPs of the BVSearch algorithms can be found. As can be seen from the specifications in Table 5.4 the quantum circuits are very short with respect to latency. Also, the number added SWAPs is very limited. The two-qubit gates have a high level of locality due to the limited number of routing events. Therefore most of the routing could probably be improved with a more optimal initial placement.

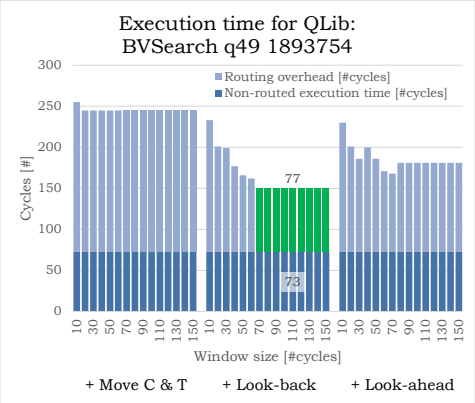


Figure 5.42: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: BVSearch 49-qubit number 1893754 algorithm.

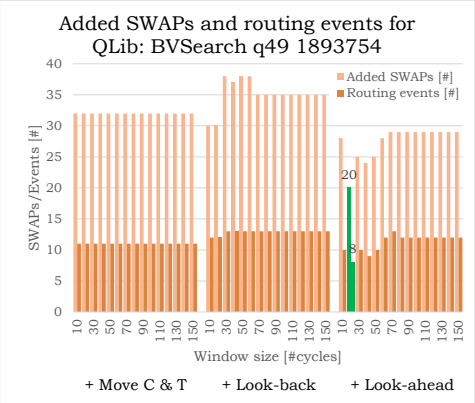


Figure 5.43: Number of routing events and added SWAPs for different functions and window size for the QLib: BVSearch 49-qubit number 1893754 algorithm.

In Figures 5.44 and 5.45, and the figures in Appendix E, the execution time and added SWAPs of the Cuccaro Adder algorithms can be found. Again the quantum circuits are short concerning latency, see the specifications in Table 5.4. Nevertheless, the routing algorithm reduces the overhead from 110% to 36% when evaluating the results from Figure 5.44 (compared moving target and control with look-ahead). This improvement reduces the total execution time with 35.4%. From the added SWAP figures, the results as found with the random benchmarks hold. The effect of look-ahead becomes very well visible. There is an ideal point where a certain window size will give a lower number of added SWAPs.

Compared to the random benchmarks with the same percentage of two-qubit gates, the look-ahead function will in this situation spread the load more evenly around all the qubits. This can be seen in the heat-maps from Figures 5.46 and 5.47. These heat-maps visualize that these algorithms have a higher qubit locality compared to the random benchmarks, due to the different 'hot-spots' in the maps. The qubits seem to cluster to minimize the overhead.

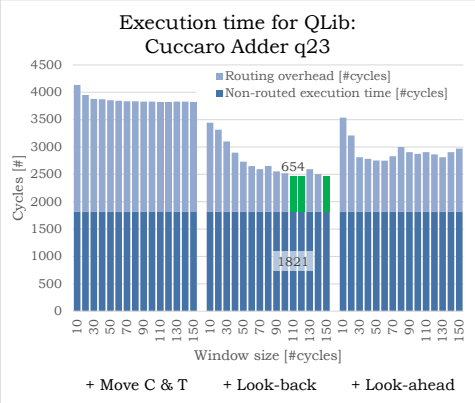


Figure 5.44: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: Cuccaro Adder 23-qubit algorithm.

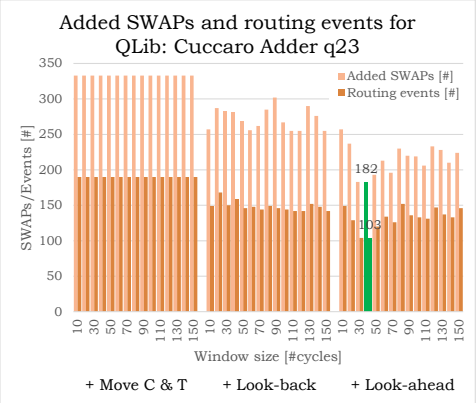


Figure 5.45: Number of routing events and added SWAPs for different functions and window size for the QLib: Cuccaro Adder 23-qubit algorithm.

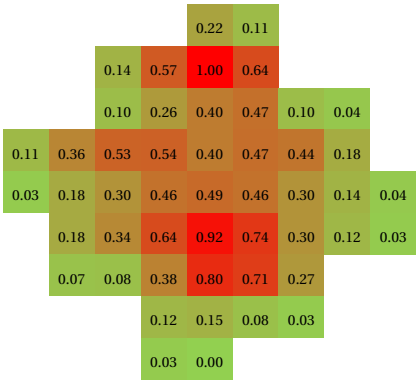


Figure 5.46: heat-map: QLib: Cuccaro Adder 23-qubit algorithm (window size 40) with look-back and without look-ahead

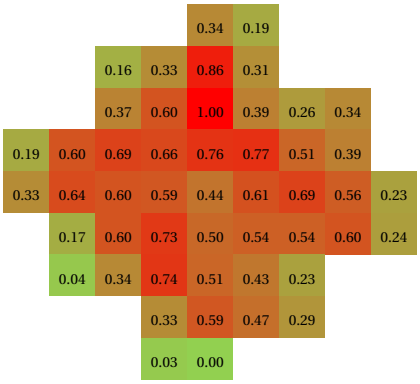


Figure 5.47: heat-map: QLib: Cuccaro Adder 23-qubit algorithm (window size 40) with look-back and look-ahead

In Figures 5.48 and 5.49, and the figures in Appendix E, the execution time and added SWAPs of the Multiplier algorithm can be found. The routing algorithm reduces the overhead from 142% to 43% in Figure 5.48 (compared moving target and control with look-ahead), which reduces the total execution time with 40.7%. In these quantum circuits, the number of instructions is much higher compared to BVSearch and Cuccaro Adder algorithms. The results are therefore more stable with different window sizes due to the more balanced number of instructions. These graphs give no new insight compared to the other quantum algorithms.

In Figures 5.50 and 5.51, and the figures in Appendix E, the execution time and added SWAPs of the QME algorithm can be found. The routing algorithm reduces the overhead from 112% to 40% in Figure 5.50 (compared moving target and control with look-ahead), which reduces the total execution time with 34.2%. In these quantum circuits, the number of instructions is even higher compared to the Multiplier algorithms. The results are therefore again more stable with different window sizes due to the more balanced number of instructions. These graphs give no new insight compared to the other quantum algorithms.

Concluding the QLib benchmarks, are the VBE Adder algorithms. In Figures 5.52 and 5.53, and the figures in Appendix E, the execution time and added SWAPs of the VBE adder algorithm can be found. In these quantum circuits, the number of instructions is comparable to BVSearch and Cuccaro Adder. The results are also comparable with the BVSearch and Cuccaro Adder. The reduction in overhead is roughly the same, in Figure 5.52 the overhead is reduced from 127% to 37% (compared moving target and control with look-ahead), which reduces the total execution time with 39.7%. These graphs give therefore no new insight compared to the other quantum algorithms.

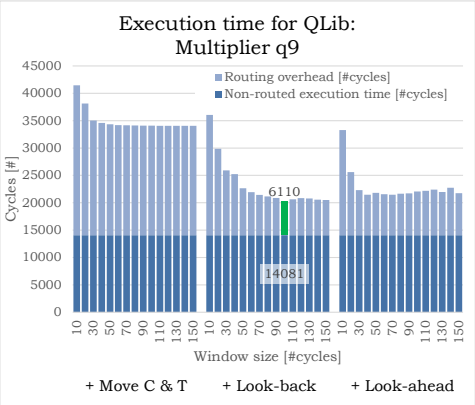


Figure 5.48: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: Multiplier 9-qubit algorithm.

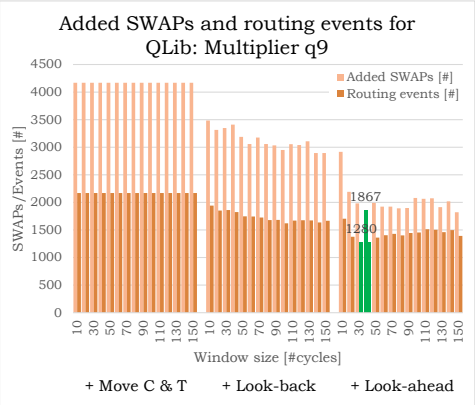


Figure 5.49: Number of routing events and added SWAPs for different functions and window size for the QLib: Multiplier 9-qubit algorithm.

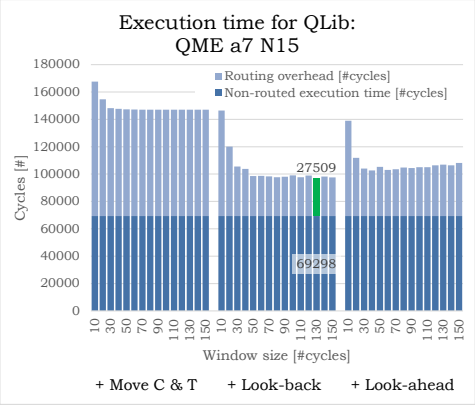


Figure 5.50: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: QME a7 N15 algorithm.

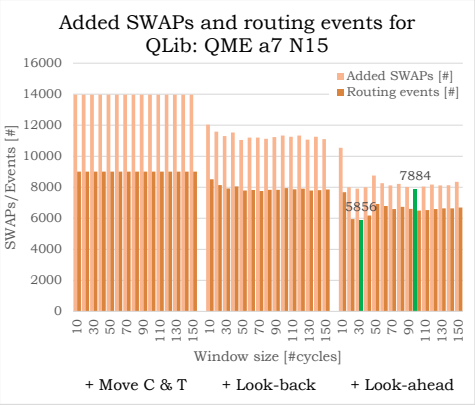


Figure 5.51: Number of routing events and added SWAPs for different functions and window size for the QLib: QME a7 N15 algorithm.

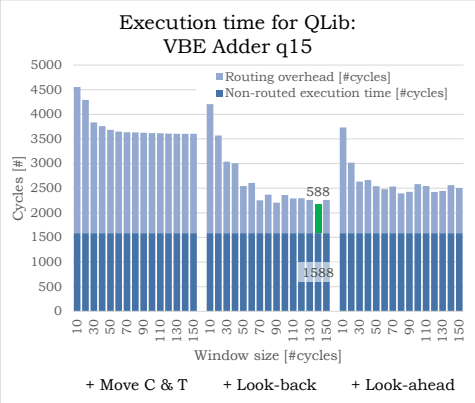


Figure 5.52: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: VBE Adder 15-qubit algorithm.

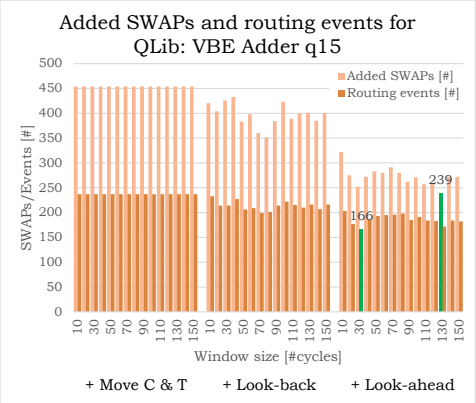


Figure 5.53: Number of routing events and added SWAPs for different functions and window size for the QLib: VBE Adder 15-qubit algorithm.

5.3. Routing planar-based logical qubits

QLib benchmarks are used to evaluate the routing algorithm for planar-based logical qubits. In this situation, the quantum algorithms are mapped on a minimal square lattice. The quantum circuits have an initial placement based on the principle of minimizing the Manhattan distance. Only the Adder, BVSearch and Multiplier are mapped. Due to the initial placement, the BVSearch does not need any routing. Therefore only the Adder and Multiplier algorithms are evaluated. The specifications of both algorithms can be found in Table 5.5.

Table 5.5: Specifications QLib benchmarks.

Benchmark		**Gates [#]**	**Two-qubit gates [%]**	**Qubits [#]**	**Lattice size**
Adder	1-8	286	44.6	25	5x5
	1-16	577	44.5	36	6x6
Multiplier	q4	1655	43.6	25	5x5

In Figures 5.54 and 5.57 the results are shown of the two QLib Adder algorithms. The variance between the different window sizes is very high. There is less consistency in the results. For example, the larger window size is more beneficial when look-back is enabled. This inconsistency probably has to do with the fact that the longest gate is not the SWAP gate anymore, see Table 4.3. Therefore avoiding logical qubits that for example perform a T gate could potentially give a lower latency. Evaluating the occupancy of future gates will become more relevant what is not taken into account in this routing algorithm. Nevertheless, the routing algorithm reduces the overhead from 43% to 10% in Figure 5.57 (compared moving target and control with look-ahead), which reduces the total execution time with 23.2%.

In Figures 5.58 and 5.59 one multiplier QLib algorithm results are shown. Although the results are a bit more consistent compared to the adder, the most optimal window size does not converge to a single point. The routing algorithm reduces the overhead from 60% to 10% (compared moving target and control with look-ahead), which reduces the total execution time with 30.9%.

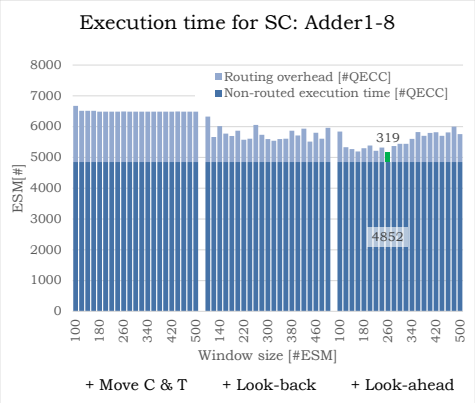


Figure 5.54: Execution time in QECCs for different functions and window sizes for the qLib: Adder1-8 algorithm.

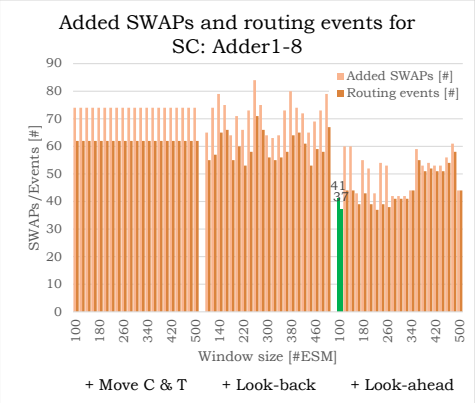


Figure 5.55: Number of routing events and added SWAPs for different functions and window size for the QLib: Adder1-8 algorithm.

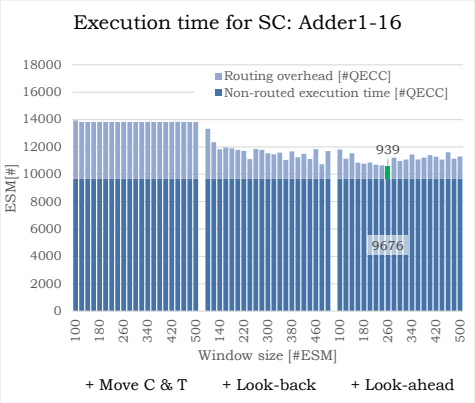


Figure 5.56: Execution time in QECCs for different functions and window sizes for the QLib: Adder1-16 algorithm.

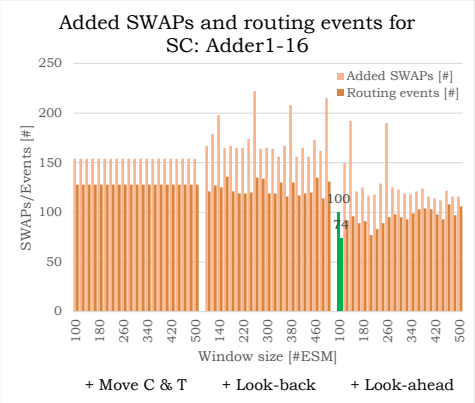


Figure 5.57: Number of routing events and added SWAPs for different functions and window size for the QLib: Adder1-16 algorithm.

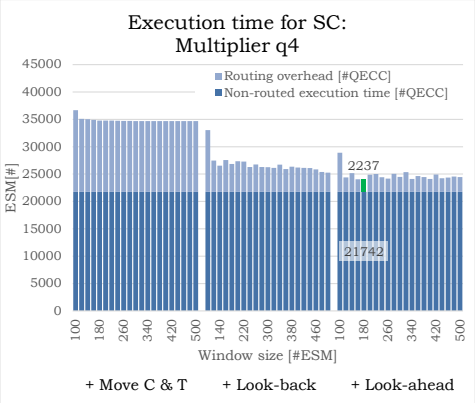


Figure 5.58: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: Multiplier 4-qubit algorithm.

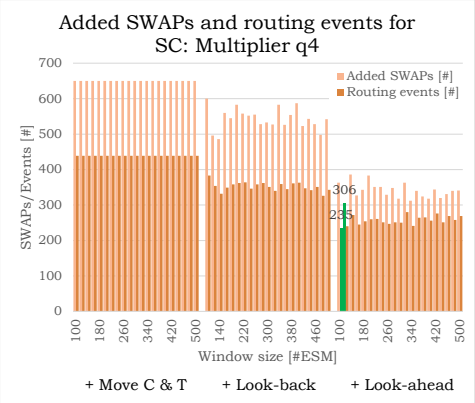


Figure 5.59: Number of routing events and added SWAPs for different functions and window size for the QLib: Multiplier 4-qubit algorithm.

5.4. Comparison with other routing algorithms

To evaluate how well the proposed routing algorithm performs, it can be compared to routing algorithms in other literature. Unfortunately as explained in Subsection 3.3.2 the related literature tries to optimize the number of inserted SWAPs and not the actual latency of the circuits. With that also the given lattice shapes are not consistent as well as used benchmarks. Therefore no direct relation can be made to the related literature mentioned in Subsection 3.3.2.

With the results from chapter 5 some conclusions can be drawn when comparing to other literature. For example, only inserting the shortest path, with moving control and target qubit, based on the number of SWAPs, will not always give the best results. When trying to interleave with previous instructions and look at future instructions, shorter path concerning the number of cycles may become visible. Therefore an evaluation only based on SWAPs will not always result in the shortest latency.

Also, the possibility to implement the routing algorithm as run-time routing algorithm may be beneficial for future large-scale quantum computers. Therefore some related literature, which implements any shape of a sliding window, uses these approaches. Again, the results from other literature are optimized on the number of SWAPs. This optimization does not need to interleave with any other quantum gate and could potentially give the opportunity to remove the entire buffer, not taking into account rescheduling, if the compiler, for example, generates opp-codes where future gates are mentioned.

6

Scalability and implementability of the routing algorithm

In this chapter, the scalability and implementability of the routing algorithm are discussed. In Section 6.1 the actual execution time of the routing algorithm itself is evaluated. With this execution time and the routing results from chapter 5 the evaluation is made if and how a routing algorithm can be used for real runtime routing, and if the routing algorithm is scalable, in Section 6.2.

6.1. Routing algorithm execution time

To calculate the execution time of the routing algorithm, the random generated benchmarks from Section 5.1 are used to perform timing measurements. Based on these executions, some conclusions can be drawn in Section 6.2 to see if this proposed routing algorithm is scalable for a larger number of qubits and if the routing algorithm is implementable at runtime.

The routing algorithm is executed on an Intel i7-6700HQ CPU @ 2.60GHz processor. Because the routing algorithm is written in Python and not multi-threaded, the processor is not fully utilized. Running the routing algorithm script on the Intel i7 processor will utilize the CPU roughly by 15%. Nevertheless, the scalability of this routing algorithm can be evaluated with some timing results.

First, each function of the routing algorithm is evaluated by the percentage it contributes to the total execution time. In Figure 6.1 and the figures in Appendix F, the total execution time for different percentage two-qubit gates (the different bars) at different window sizes is shown. The percentage of the total execution time of each component is shown in these figures. It can be seen from these figures that the rescheduling block is the most time-consuming operation. Depending on the percentage two-qubit gates and the window size, the rescheduling part uses roughly 65% up to 90% of the execution time. Because this process inside the routing algorithm is probably far from ideal, the execution time without rescheduling is plotted in Figures 6.2 and the figures in Appendix F. In these figures, some functions like splitting parallel instructions are not visible due to there nil contribution to the total execution time. From these figures, it is clear that the look-back and look-ahead times grow with the increasing percentage two-qubit gates, roughly 55% up to 90% of the execution time excluding rescheduling. These growing times have to do with the fact that with increasing two-qubit gate percentage more routing events are occurring. Therefore, the total amount both functions are required increases. The different window sizes will also increase the total execution time, as well as the contribution of the look-back and look-ahead function because more gates are taken into account when using these functions.

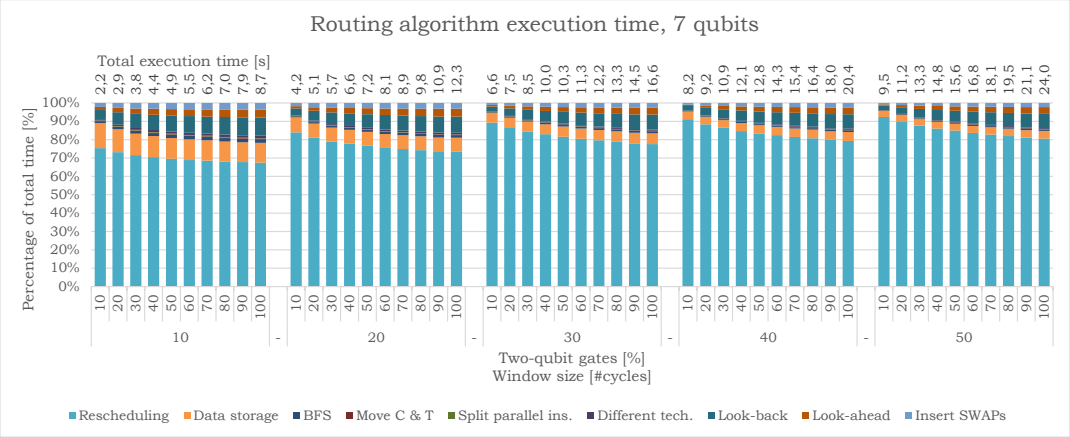


Figure 6.1: The execution time of the routing algorithm for a 10000, 7 qubit, a random generated circuit for different percentage of two-qubit gates, 10-100% (the different bars) for a window size of 10-50[#cycles]. The contribution (in percentage) to the total time of the different functions is shown.

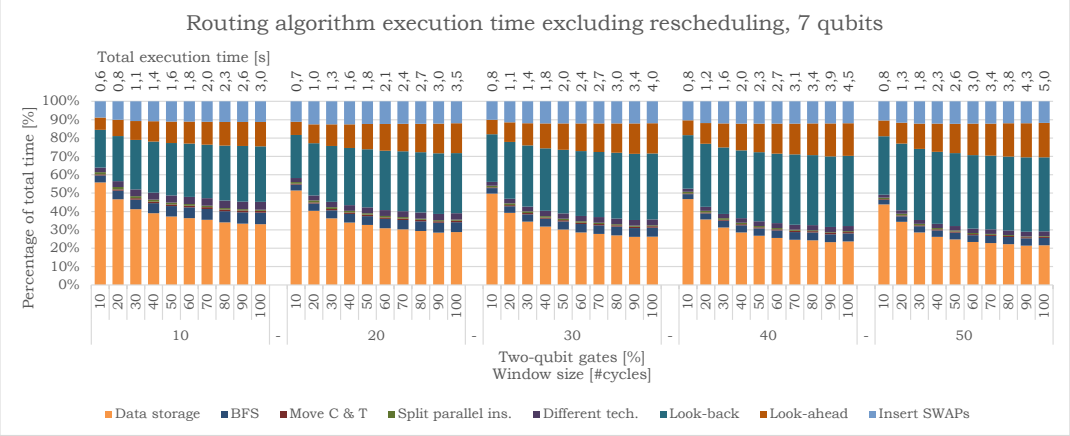


Figure 6.2: The execution time of the routing algorithm for a 10000, 7 qubit, a random generated circuit for different percentage of two-qubit gates, 10-100% (the different bars) for a window size of 10-50[#cycles]. The contribution (in percentage) to the total time excluding rescheduling of the different functions is shown.

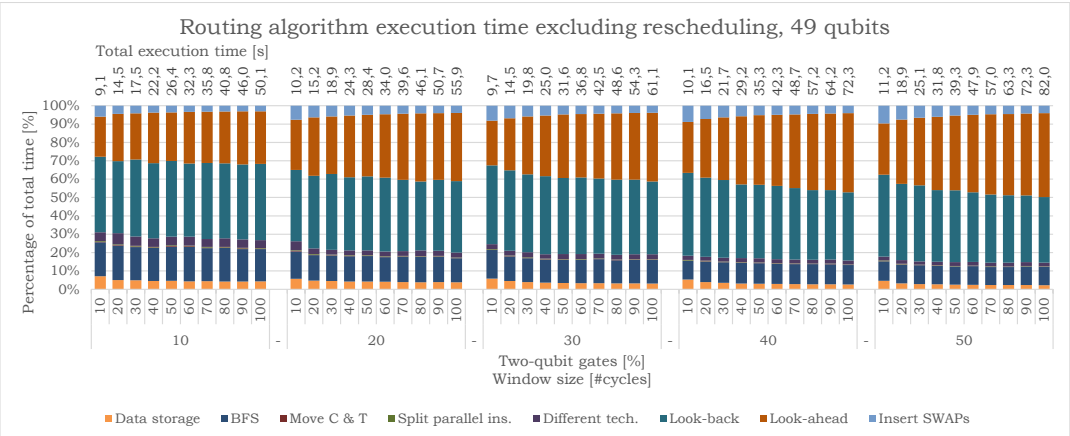


Figure 6.3: The execution time of the routing algorithm for a 10000, 49 qubit, a random generated circuit for different percentage of two-qubit gates, 10-100% (the different bars) for a window size of 10-50[#cycles]. The contribution (in percentage) to the total time excluding rescheduling of the different functions is shown.

When looking at the 49 qubits random circuits in Figure 6.3, and in Appendix F the same conclusions hold as for the 7 qubits random circuits. But when increasing the number of qubits inside the algorithm mainly the BFS, look-ahead, and look-back functions are increasing in contribution to the total time. This contribution can be simply clarified by the fact that when increasing the lattice, the larger number and higher variety of paths should be evaluated, and also more qubits are affected by those paths.

To summarize, the total execution time with and without rescheduling for the different quantum circuit qubit sizes and percentage two-qubit gates are shown in Figure 6.4 and 6.5 respectively. In these figures, the different trends for one type of qubits are the different percentage two-qubit gates. From these figures can be concluded that increasing the total number of qubits will have an exponential growth in the total routing algorithm time. The percentage two-qubit gates will have a linear contribution to the execution time when a full random quantum circuit is used.

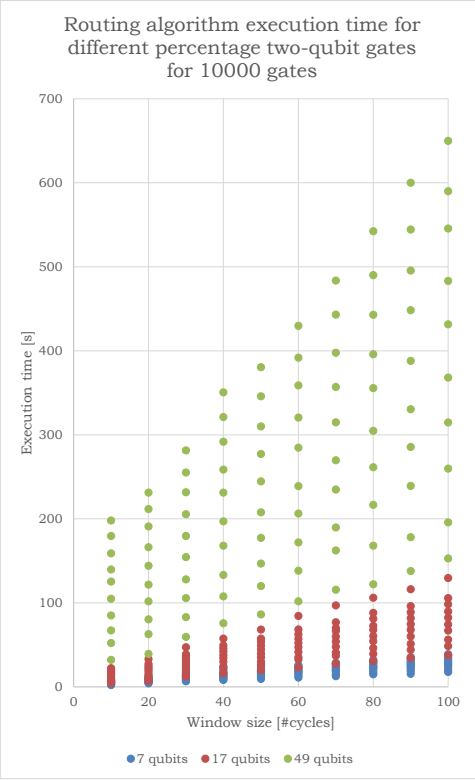


Figure 6.4: The total execution time of the routing algorithm for a different number of qubits, percentage two-qubit gates and window sizes. The different lines are the different percentage two-qubit gate. The higher the line, the higher the percentage two-qubit gates.

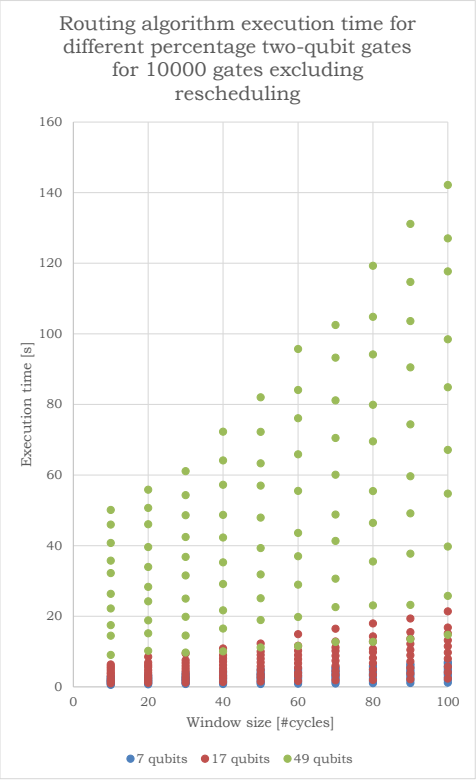


Figure 6.5: The total execution time, excluding rescheduling, of the routing algorithm for a different number of qubits, percentage two-qubit gates and window sizes. The different lines are the different percentage two-qubit gate. The higher the line, the higher the percentage two-qubit gates.

6.2. Routing algorithm implementation and scalability

The routing algorithm is designed in a way that not all the instructions are needed at the moment of routing by using a sliding window. Therefore the principle of this routing algorithm can be implemented at runtime. For executing the quantum circuit at runtime, a buffer (the sliding window) is needed. Inside this buffer, the qubits are mapped to the underlying qubit technology where the NN constraints apply. By inserting one path at a time, a constant flow of instructions can be written to the output or executed directly on the qubit plane.

When evaluating the different figures, for example Figure 6.4 and 6.5, the proposed routing algorithm is not very scalable. When increasing the number of qubits, the execution time grows exponentially. This increase in execution time when increasing the number of qubits is also a problem for runtime routing. The current version of the routing algorithm is far too slow. Due to the ns execution time of single instructions in

the current transmon technology, the necessary speed-up of this routing algorithm should be at least $1,000x$ including rescheduling. Therefore not only the rescheduling should be improved, but also the other parts of the routing algorithm. This speed-up can be achieved in multiple ways.

Due to BFS, all the possible shortest paths are evaluated. When decreasing the number of paths, the rest of the functions will also speed-up as less evaluation has to be done. The speed-up of the routing algorithm can be done by using dedicated paths instead of using BFS to find all the possible paths. Paths based on the classical routing algorithm as explained in Section 3.2 may be useful. When using these classical routing algorithms, a small selection of paths and congestion evaluation can be used to find and insert the most optimal path. How well the new routing algorithm performs can be compared with the routing algorithm proposed in this thesis where all paths are evaluated. As an addition, the current version of the routing algorithm is single threaded. Therefore all the possible multi-core or in the future multi-processor computational power cannot be utilized. Different paths can be evaluated at the same time in parallel. Therefore multi-threading may speed-up the routing algorithm significantly.

In the current routing algorithm, there are also different speed-up possibilities. The look-back function can be improved by using counters to tell when the latest instruction is being completed instead of checking each time inside the buffer to see when a qubit is occupied. Also, the look-ahead function can make use of look-up tables instead of searching through the buffer. To make this even faster, the compiler can also pre-generate opp-code which tells how many gates there are in the future and what their location is. When doing so, the look-ahead space can be removed from the buffer as well.

When looking at the long-term quantum developments, where multiple planar-based logical qubits will form tiles inside the qubit plane, the basic principle of this routing algorithm may also be useful. The different functions of the algorithm support the routing of logical qubits. When in the future ancilla factories may be needed, fast adaptive run-time routing will be critical when producing those ancilla qubits in the correct state and bring them to the right location. These ancilla qubits, which may, for example, be produced as an EPR-pair or used for implementing FT ‘T’ and ‘S’ gates, currently need a nonconstant time to be produced with a high fidelity [4] [8]. These logical ancilla qubits are transported when needed, but their origin inside the compiler is unknown. The run-time routing algorithm will make sure that the correct logical ancilla qubit is brought to the correct location. An example of a future qubit lattice architecture is illustrated in Figure 6.6 where dedicated logical blocks with communication channels are used.

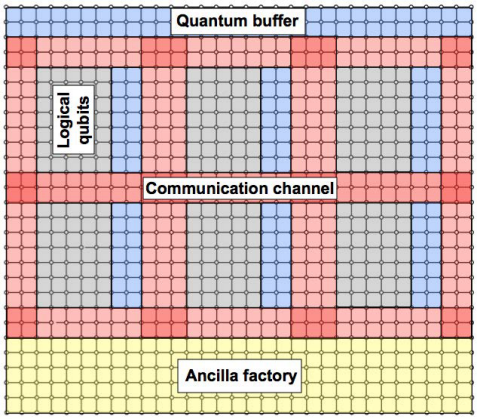


Figure 6.6: A qubit plane architecutre [8].

7

Conclusions and future work

7.1. Conclusions

In this thesis, an algorithm for routing quantum states on a 2D plane with only NN interactions is proposed. Based on a sliding window principle, the algorithm finds all possible shortest (and optional non-shortest) paths between two qubits that need to interact and are not adjacent. To this purpose, a slightly adapted classical graph search algorithm, BFS, is used. The routing algorithm tries to optimize the total latency by finding all paths with the same length and inserting the path which gives the shortest latency. To do so, it takes into account how much the instructions (SWAP gates) describing a path, can be interleaved with previous gates and tries to minimize the disturbance of qubits that need to perform a two-qubit gate in the future.

The routing algorithm will detect, inside the sliding window, if an instruction does not obey the NN constraint. If the instruction does not obey the NN constraint, a number of functions are used to find and insert the shortest path option with respect to latency in cycles. 1) The instruction, which does not obey the NN constraint, can be split from its parallel instructions. By doing so, possible extra overhead may be avoided as a path may disturb the NN qubits in the parallel gates. 2) The routing algorithm will find all path based on BFS with the same length. These paths can be split to move both control and target qubit. Each possible path and intermediate step of meeting in the middle is stored. 3) For each path, different transportation techniques are used. Due to the small latices analyzed in this thesis, only standard SWAP based transportation is used. 4) Next, each option is evaluated whether and by how many the transportation gates can be interleaved with previous instructions. 5) Finally, each option is evaluated for future gates. This evaluation is done by temporary applying the new qubit mapping, due to a selected path, to sum the minimal distance between all qubits in the future two-qubit gates. The minimal distance between all future gates is translated to a number of cycles, and add up to the path length, and the number of cycles each path may interleave is subtracted. The path with the shortest delay in number of cycles is inserted, and the instructions are rescheduled ASAP.

To see how well the routing algorithm performs, the thesis first evaluates random generated quantum circuits with a variable percentage of two-qubit gates. From these random generated quantum circuit benchmarks can be concluded that splitting parallel instructions helps to reduce communication overhead. Also more path options by allowing both qubits to move to meet in the middle, give the routing algorithm the possibility to select a faster option. Splitting the paths in moving both qubits has a significant positive effect on the overhead. Depending on the number of qubits, the absolute reduction in overhead varies from 50% for 7 qubits with a low percentage of two-qubit gates, till over 200% for 17 qubits with a high percentage two-qubit gates. The window size has a significant influence on the overall performance of the routing algorithm. A larger window size has a positive effect when concerning the rescheduling of the quantum circuit. If the window is larger, more operations can potentially be rescheduled more efficiently. This rescheduling effect deludes when increasing the window size, as at the end it depends on the dependency with previous instructions if the rescheduling can reschedule those instructions. The functions that take into account the other instructions (look-back and look-ahead) have an advantage on the overhead. An added advantage on top of the previously mentioned percentage with a reduction in percentage, varying from 50% for 7 qubits with a low percentage of two-qubit gates, till over 500% for 49 qubits with a high percentage two-qubit gates.

When evaluating the random benchmarks, the window size plays an important role when a path is selected based on the other qubit instructions. The look-back function can potentially interleave the paths with earlier instructions if a larger sliding window is selected. Although this effect is limited till the last moment, the previous qubits are occupied. When a qubit is occupied in earlier instructions, having a larger sliding window will not give any benefit. The look-ahead functions evaluate the disorder of the future two-qubit gates when inserting a specific path. If the window size is larger and/or the quantum circuits contain a higher percentage two-qubit gates, more two-qubit gates are evaluated. The principle of evaluating the distance between future qubits in the short term can potentially have a positive effect on the overall quantum circuit duration. When an increasing number of future two-qubit gates are evaluated, a path is selected based on all two-qubit gates. But two-qubit gates which are far away in the future, especially when the future two-qubit gates have no locality, will potentially be relocated to a new position when more paths are inserted. Therefore looking into the far future does not make sense, because the near future will potential reorder the qubits anyway. To conclude the current look-ahead function, the function does always produce a lower number of inserted SWAPs but does not always produce a shorter quantum circuit concerning execution time.

The look-ahead function tries to minimize the number of SWAPs. This minimization is effective, concerning latency, for a low number of qubits or percentage of two-qubit gates, but less efficient when an opposite type of quantum circuit is routed. When looking at the number of added SWAPs, and consequently the total number of gates, look-ahead will always be beneficial. When look-ahead to future instructions, different variables play an important role; the percentage two-qubit gates, the locality of the two-qubit gates, and the distribution of two-qubit gates inside the quantum circuit. If there is any locality of the two-qubit gates, there is the possibility to find a path avoiding the 'hot-spots'. The distribution of gates can potentially lead to extra delay when all qubits need to be routed at the same time. When using random benchmarks, the percentage two-qubit gates can be controlled, there is no locality, and the distribution of the two-qubit gates is constant. To improve the quality of the look-ahead function, these dependencies should be evaluated by selecting a fixed number of future two-qubit gates which are operating in the same area.

When going to real quantum benchmarks from ScaffCC and QLib, the distribution and the locality of qubits do vary inside these benchmarks. Due to the more structured way the quantum circuit is built some results are slightly different compared to the random generated benchmarks. All the functions, except for look-ahead, have the same effect as previously mentioned. Due to the locality and distribution of gates inside of the different benchmark, the look-ahead function can produce different results depending on the window size. The most optimal point with respect to the window size is sometimes more narrow, but this all depends on the actual quantum circuit.

Finally, the routing algorithm is also used to map a quantum circuit based on planar-based logical qubits. With the limited availability of benchmarks, the results from the physical qubit implementation also hold for planar-based logical qubits. The effect of the look-ahead function compared to physical qubit routing is less visible because the SWAP gate does not have the longest duration. From these results, possible interleaving with future gates may be very beneficial when routing these quantum circuits. If a qubit is performing a longer gate operation, avoiding these qubits could potentially give a resulting lower quantum circuit latency.

7.2. Future work

When evaluating the routing algorithm proposed in this thesis some future research may be needed. Despite the fact that not all possible functions been analyzed or been evaluated, also improvements to the routing algorithm can be made.

Different shapes

In the current version of the python script, it is possible to specify different shapes of a 2D qubit plane. In this thesis, only the SC shapes for the physical qubits or minimal square lattice for the logical qubits are evaluated. Therefore it would be interesting to evaluate different shapes and sizes, for example, 1D or ring shapes, for the same benchmarks.

Different transportation techniques

Due to the small number of qubits only the SWAP based transportation techniques where relevant. When changing the lattice shapes or increasing the number of qubits the path length may increase. With this increasing path length, the different transportation techniques explained in Section 3.4 may be more beneficial. When adding those techniques in the routing algorithm, shorter routing solutions concerning latency may be selected.

Also related to the different transportation techniques, when routing planar-based logical qubits another transportation technique can be used which is not mentioned in Section 3.4. The so-called "MOVE" operation can be used to MOVE a qubit state to another location. This MOVE operation is not performed in a 90° angle but in a normal NN direction. The MOVE operation implies some difficulties, for example, the need of an ancilla qubit in the $|0\rangle$ or $|+\rangle$ state depending on the direction of the MOVE operation. While a normal SWAP needs six merging and splitting operations, the MOVE operation only needs one of both.

Effects of future gates:

Interleaving with future gates: The current version of the routing algorithm only looks to future gates by tracking the minimal distance between two qubits inside a gate if a certain path is selected. It is not taken into account if a qubit is occupied. Therefore the look-ahead function will probably give a better result if also qubit occupancy is taken into account.

Weight future gates: All the two-qubit gates inside the sliding window have the same contribution to the path selection. Gates which are further away may possible have less effect, and therefore their contribution to the path selection should potentially be lower. When this is taken into account, the possible path selection may also improve. Related to that, currently, if two-qubit gates in the future are executed in parallel, the delay for the minimal path is summed, while in reality these paths may be executed in parallel.

Locality and distribution future gates: In a quantum circuit, two-qubit gates can have a certain level of locality and distribution. The current implementation of the sliding window will evaluate a fixed number of instructions and does not taken into account the distribution and locality of the two-qubit gates. Instead of a sliding window, a fixed number of two-qubit gates which are located in the area and are influenced by one or more path options should be evaluated. This locality can be visualized by a square box with the target and control as the two outer corners of this box. All two-qubit gates that have one or both qubits inside this box should be taken into account by the evaluation.

Move multiple two-qubit gate instructions at the same time

The routing algorithm only finds a path for a single two-qubit gate at a time. If multiple two-qubit gates are moved at the same time, they might benefit from using, for example, the same path. This moving two-qubit gates at the same time might be done by evaluating each SWAP individually. Another option might be by finding multiple paths at the same time and try to optimize this combination of paths.

Initial placement

As can be seen in Subsection 5.2.1 an initial placement might be useful to reduce the overall latency. In this example, the qubits are located at totally the wrong initial location, and therefore extra latency is added which might have been avoided if a smarter initial placement was used.

Path selection

For evaluating the shortest path, not all paths should be taken into account. For example, moving a qubit state all the way across the lattice is probably never selected. Therefore a selection of paths can be used. Also, the different classical routing algorithms mentioned in Section 3.2 may potentially be useful when making a smaller selection of paths.

Speed-up the routing algorithm

The current Python implementation is very slow. Therefore several different steps could be done to speed-up the algorithm. Some possible improvements are mentioned in Section 6.2.

Runtime routing

The routing algorithm is based on the sliding window and could despite from the current execution time be implemented as runtime routing. There are some possible changes which may improve the possible implementation of routing algorithm at runtime. Some possible changes are mentioned in Section 6.2.

A

Routing results random generated benchmarks

A.1. 7 qubits

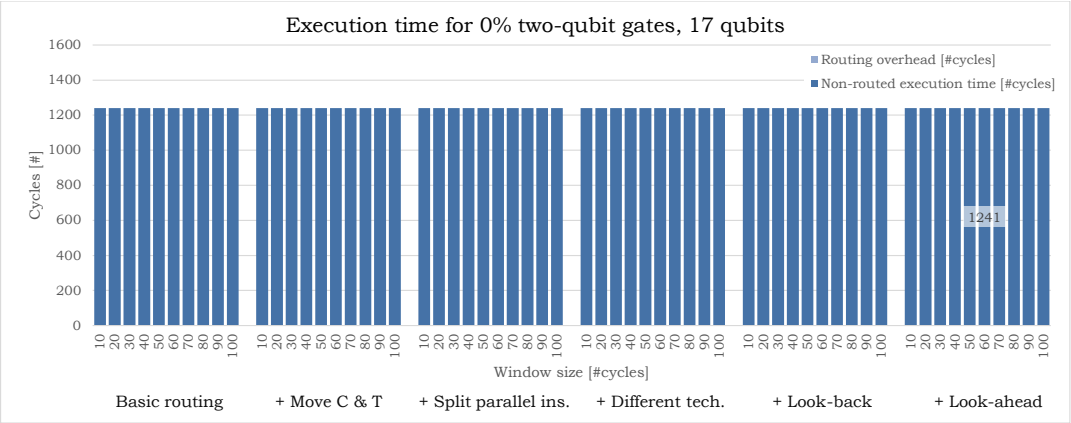


Figure A.1: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 0% two-qubit gates for 7 qubits.

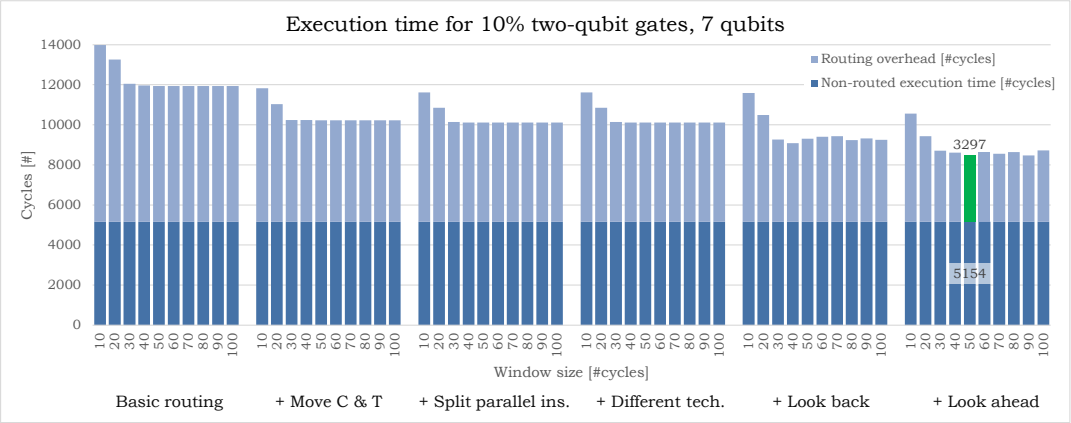


Figure A.2: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 10% two-qubit gates for 7 qubits.

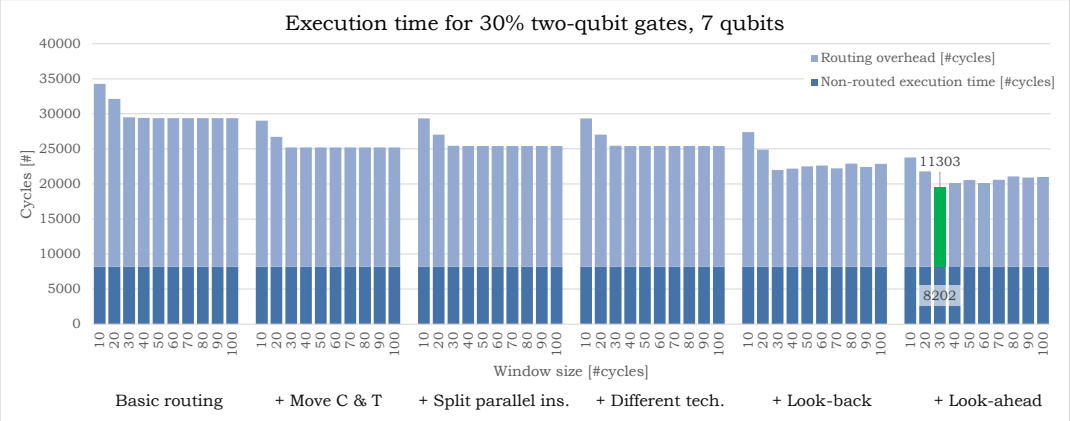


Figure A.3: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 30% two-qubit gates for 7 qubits.

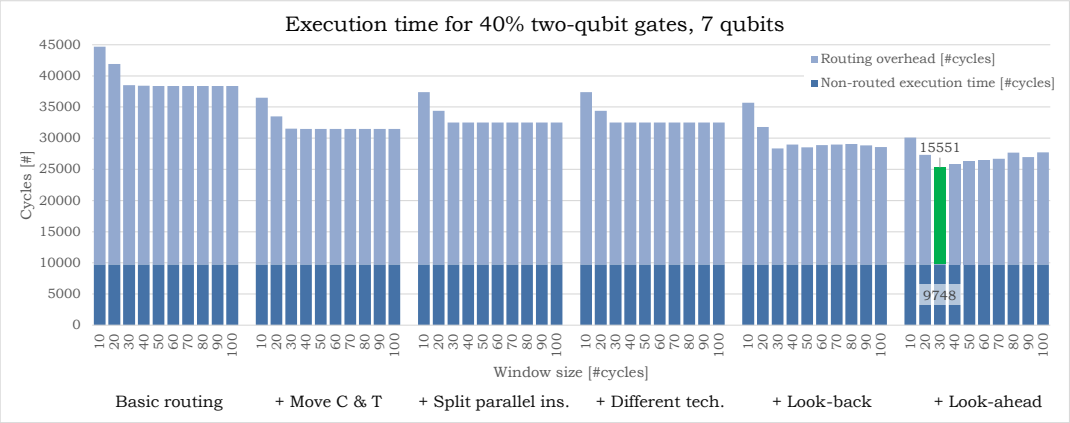


Figure A.4: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 40% two-qubit gates for 7 qubits.

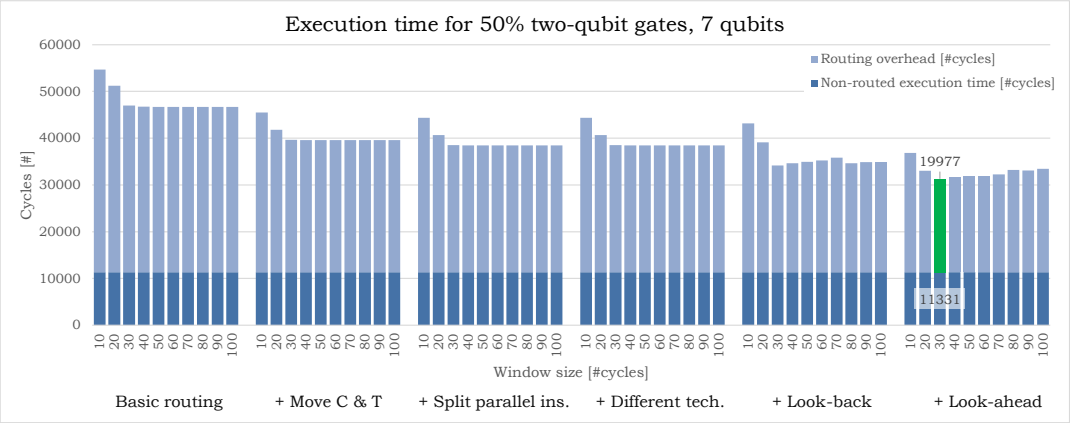


Figure A.5: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 50% two-qubit gates for 7 qubits.

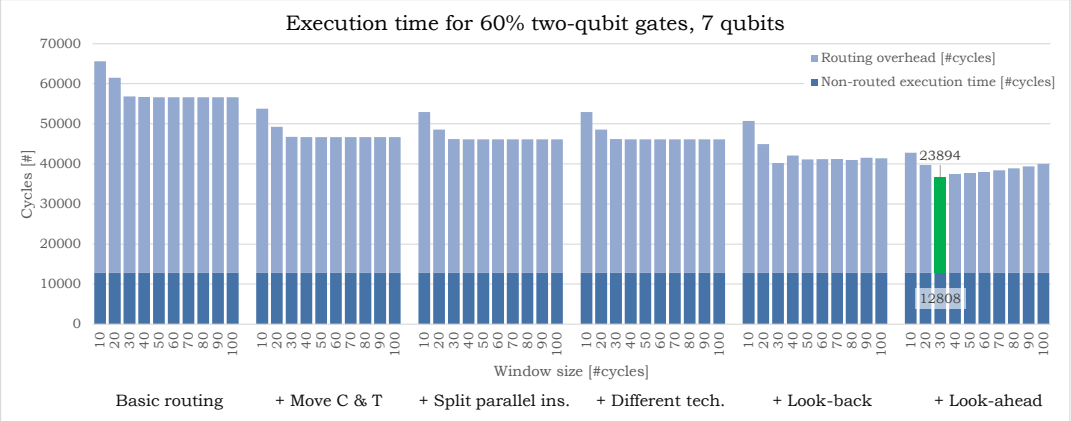


Figure A.6: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 60% two-qubit gates for 7 qubits.

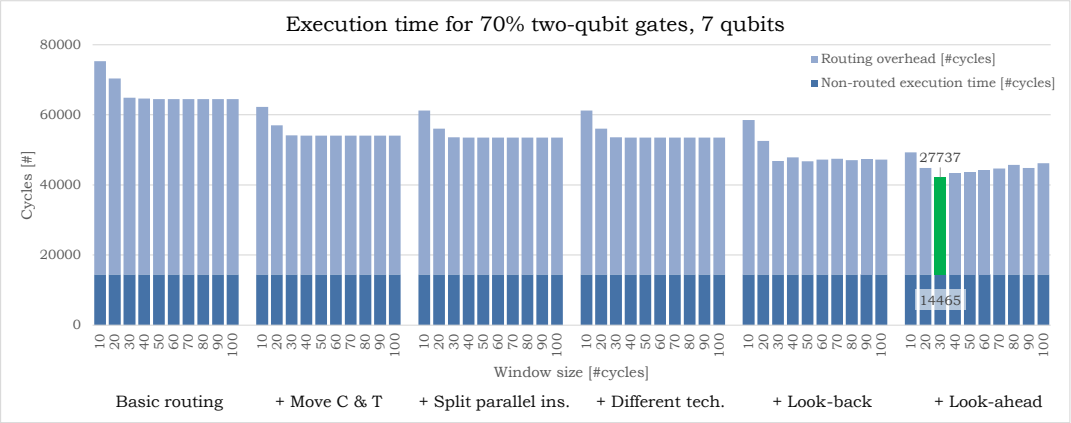


Figure A.7: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 70% two-qubit gates for 7 qubits.

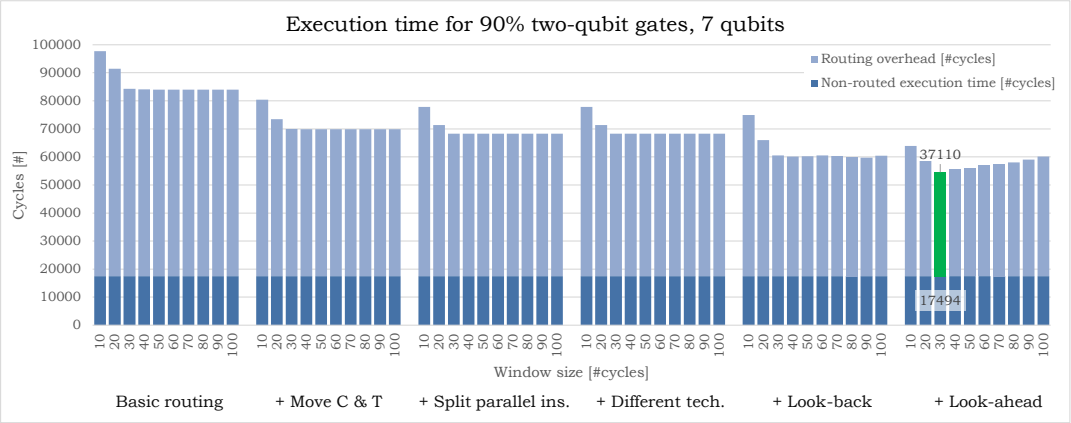


Figure A.8: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 90% two-qubit gates for 7 qubits.

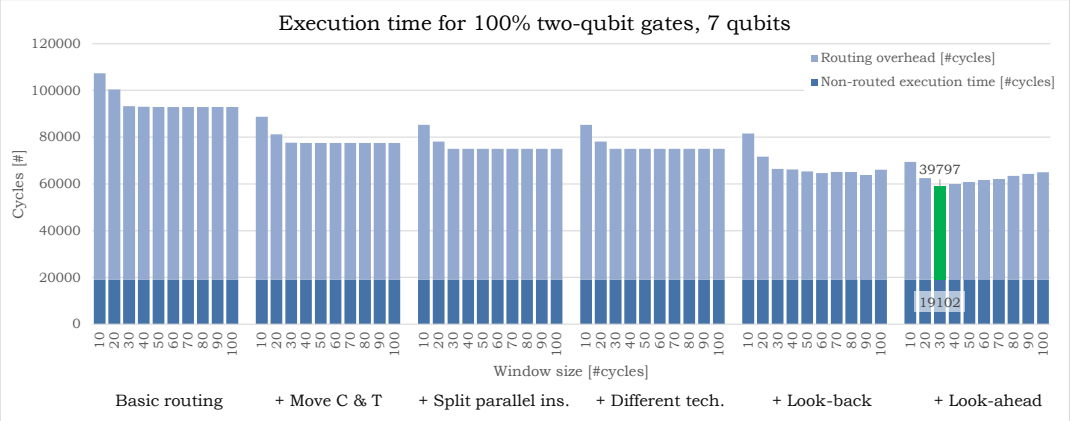


Figure A.9: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 100% two-qubit gates for 7 qubits.

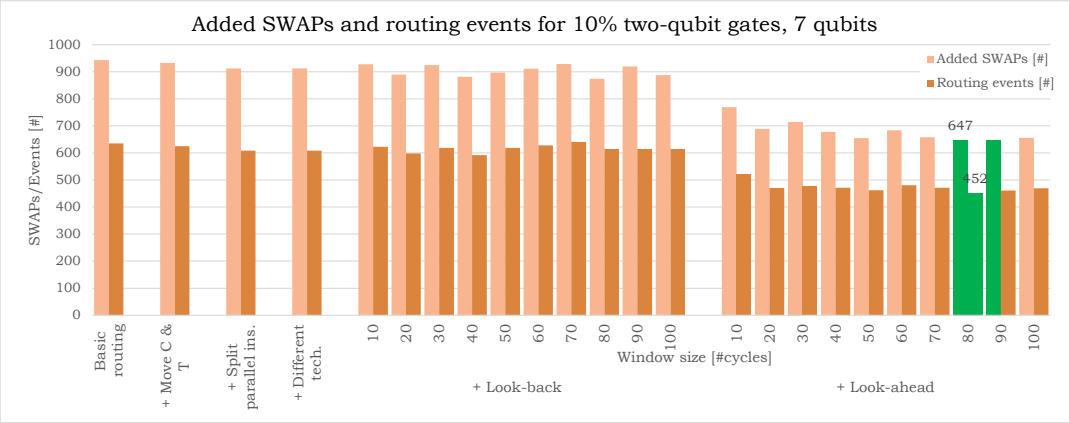


Figure A.10: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 10% two-qubit gates for 7 qubits.

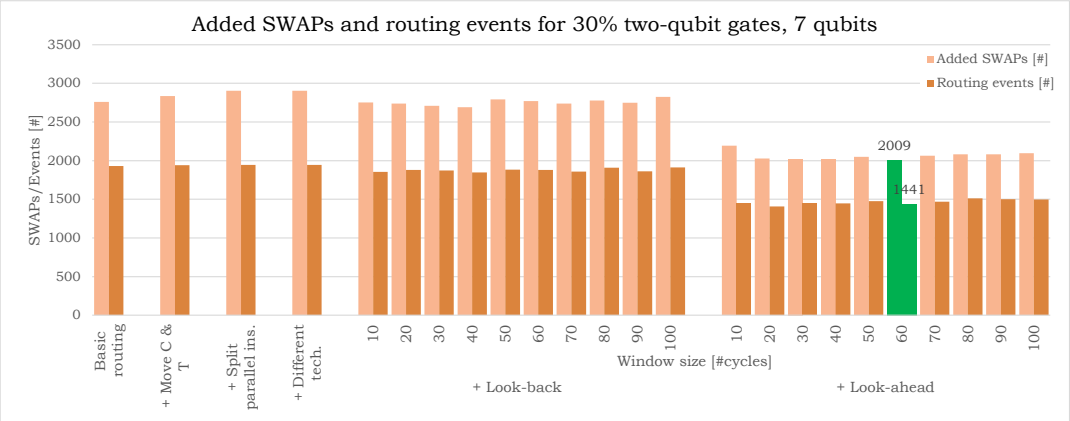


Figure A.11: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 30% two-qubit gates for 7 qubits.

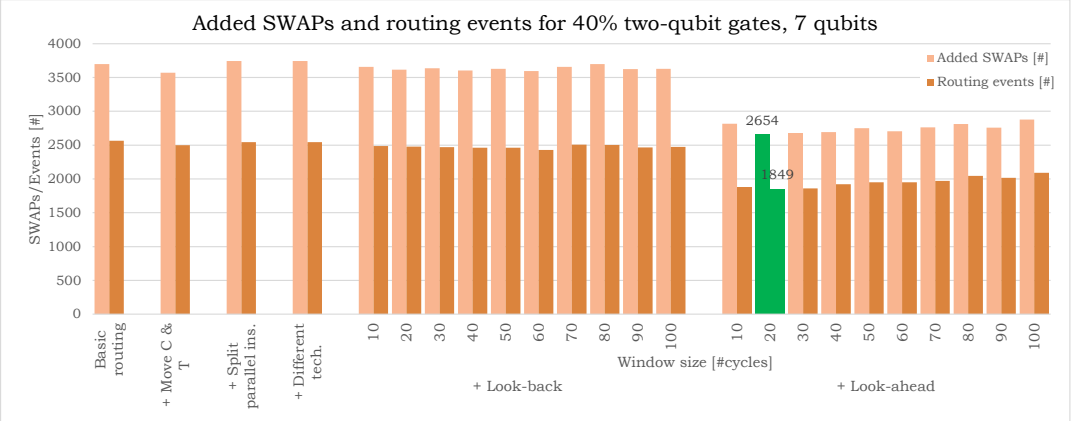


Figure A.12: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 40% two-qubit gates for 7 qubits.

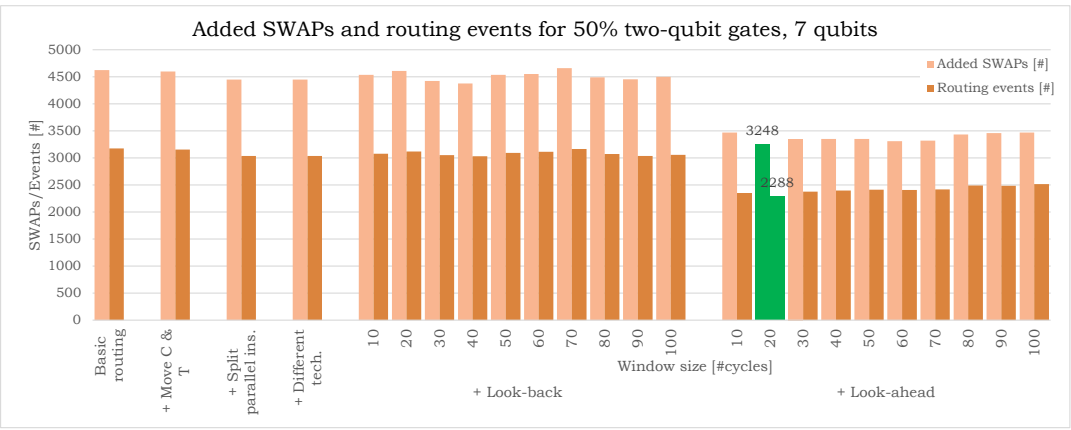


Figure A.13: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 50% two-qubit gates for 7 qubits.

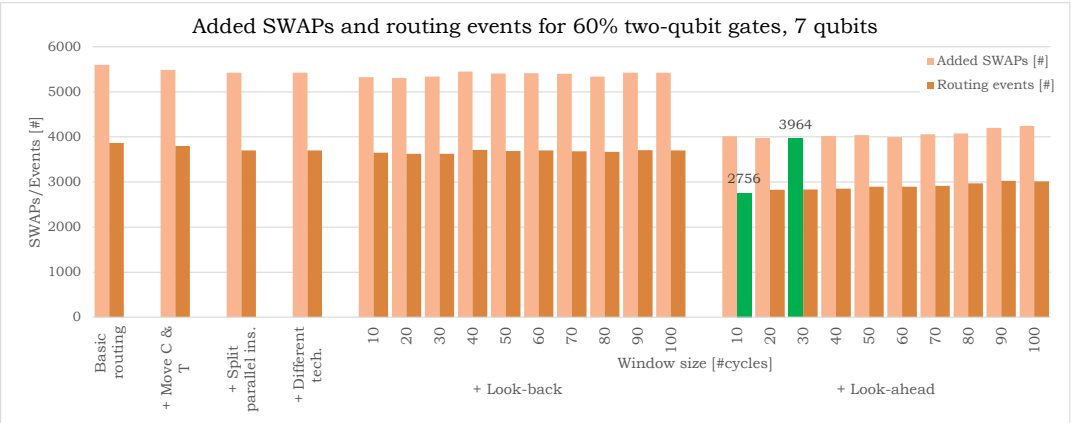


Figure A.14: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 60% two-qubit gates for 7 qubits.

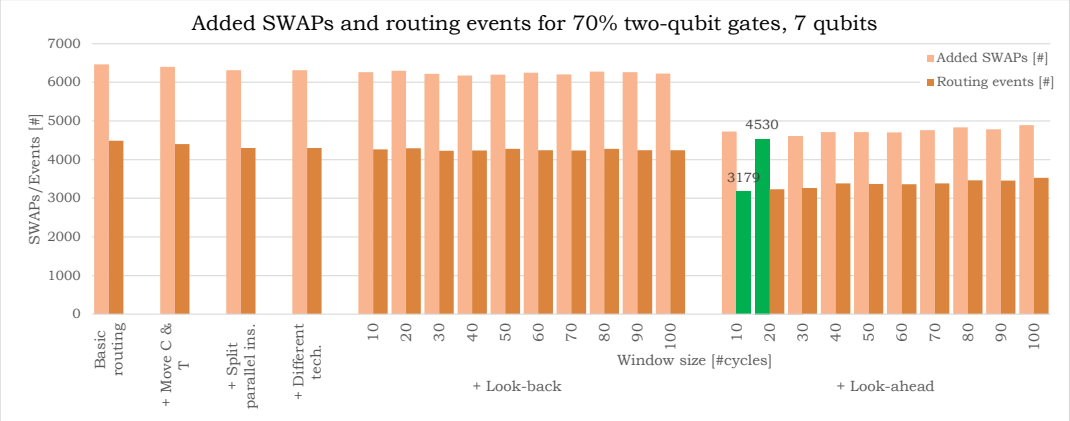


Figure A.15: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 70% two-qubit gates for 7 qubits.

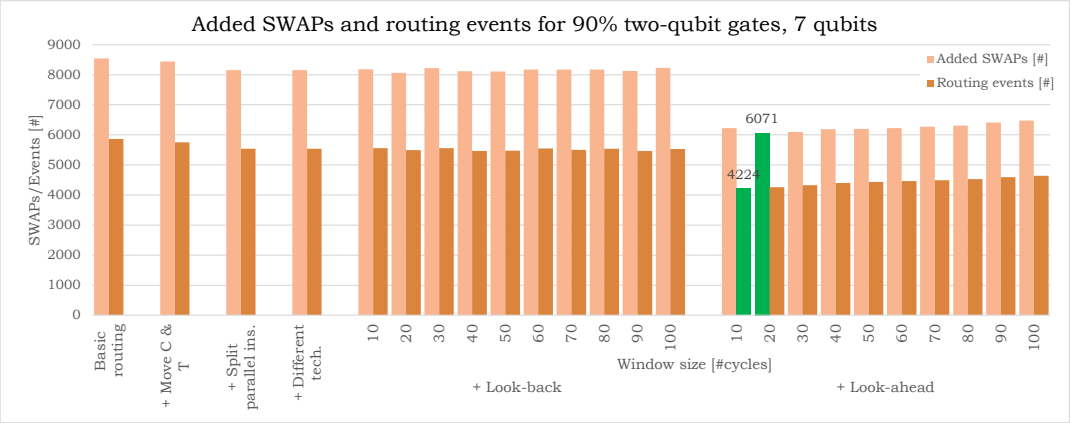


Figure A.16: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 90% two-qubit gates for 7 qubits.

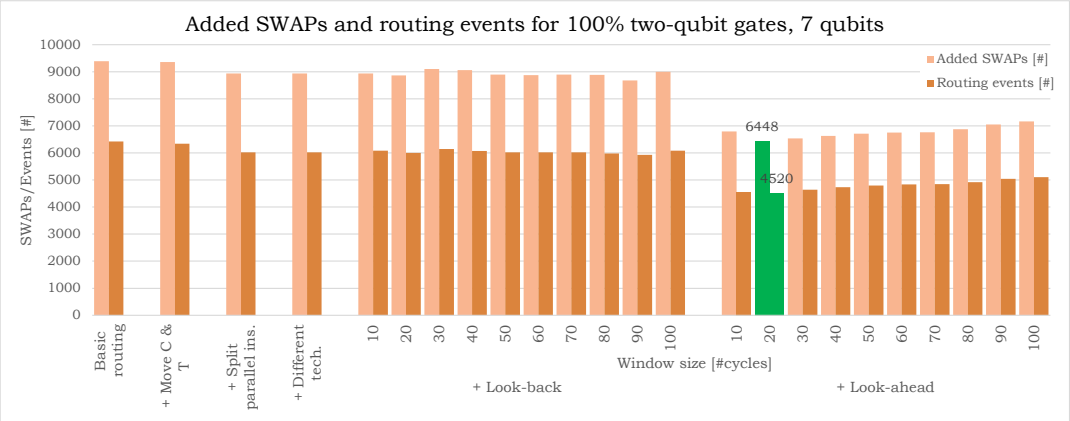


Figure A.17: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 100% two-qubit gates for 7 qubits.

A.2. 17 qubits

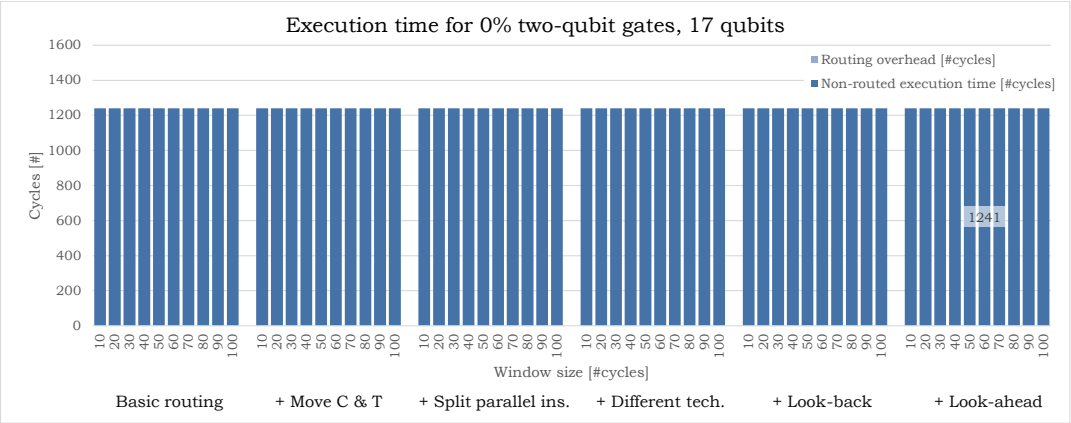


Figure A.18: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 0% two-qubit gates for 17 qubits.

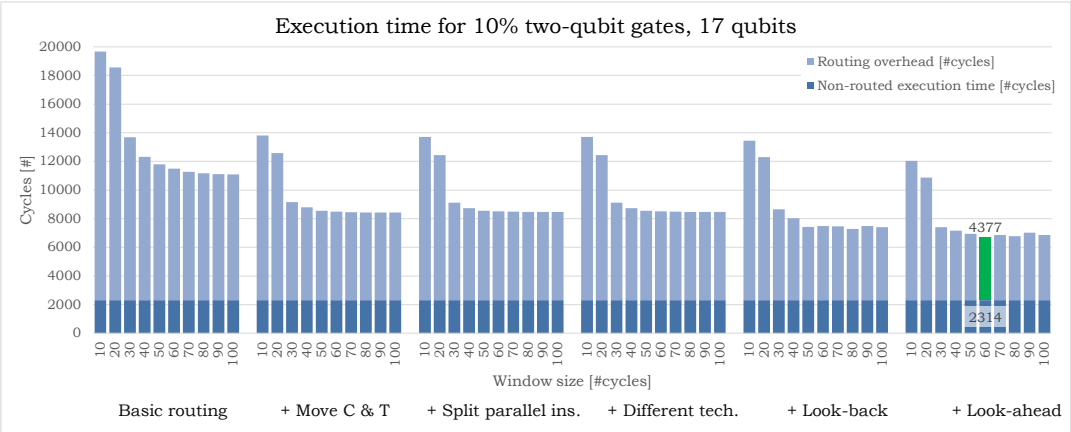


Figure A.19: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 10% two-qubit gates for 17 qubits.

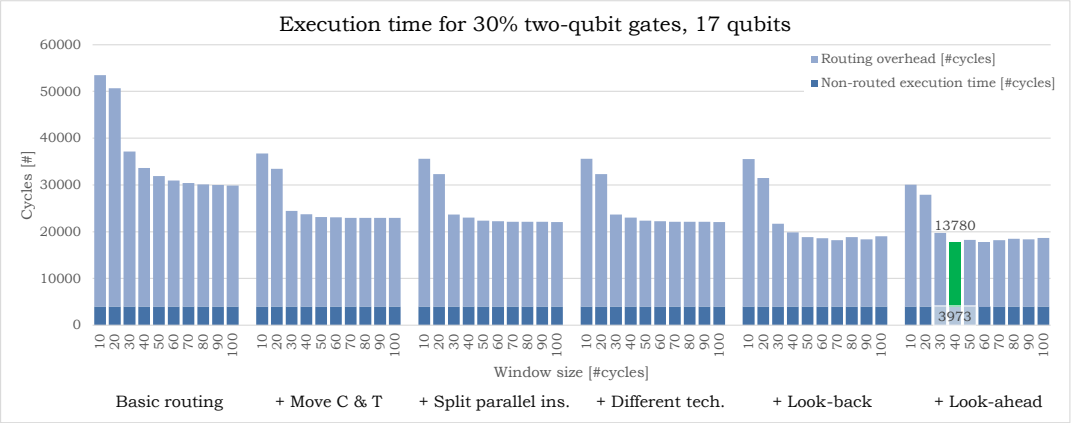


Figure A.20: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 30% two-qubit gates for 17 qubits.

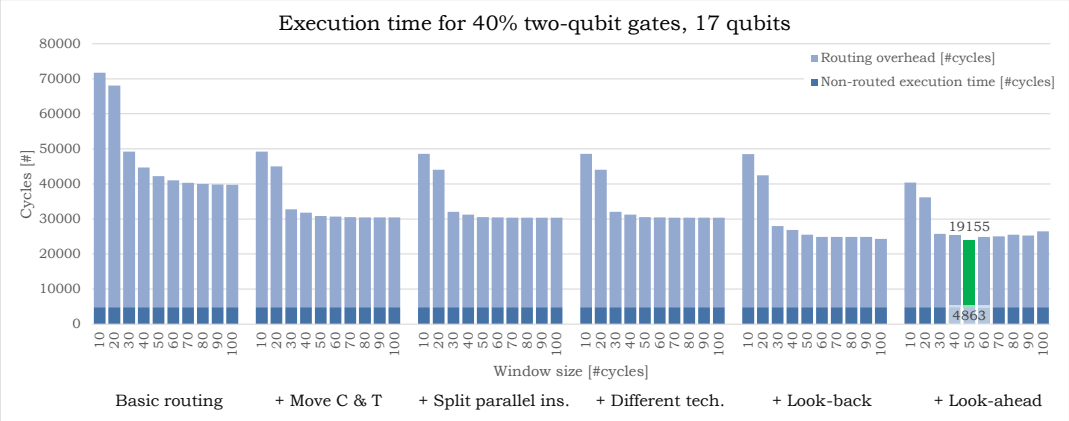


Figure A.21: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 40% two-qubit gates for 17 qubits.

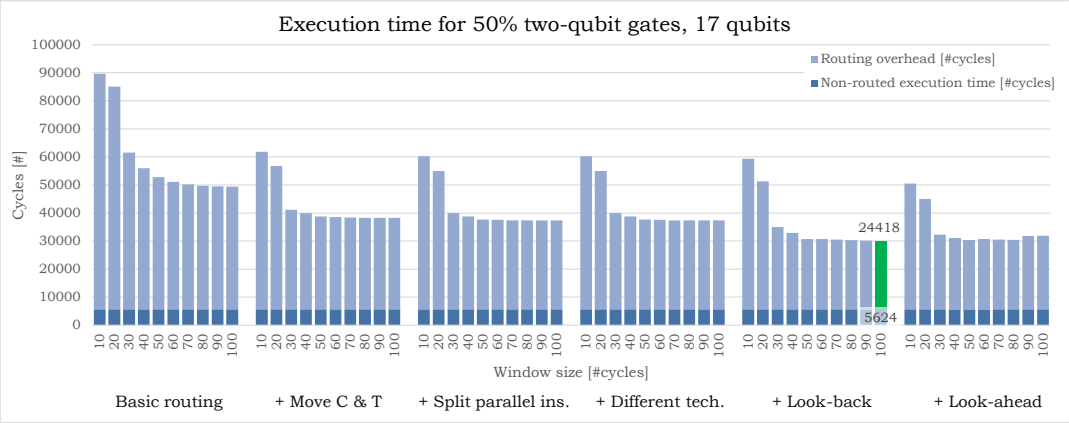


Figure A.22: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 50% two-qubit gates for 17 qubits.

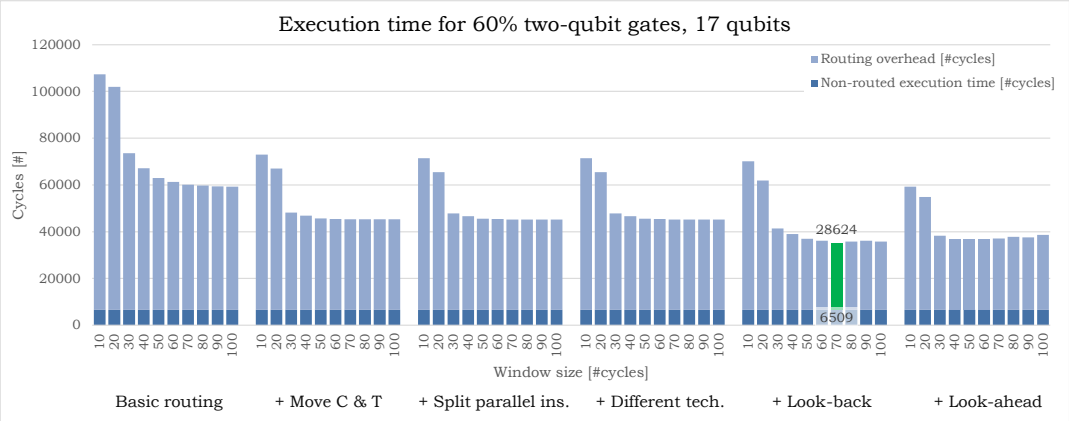


Figure A.23: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 60% two-qubit gates for 17 qubits.

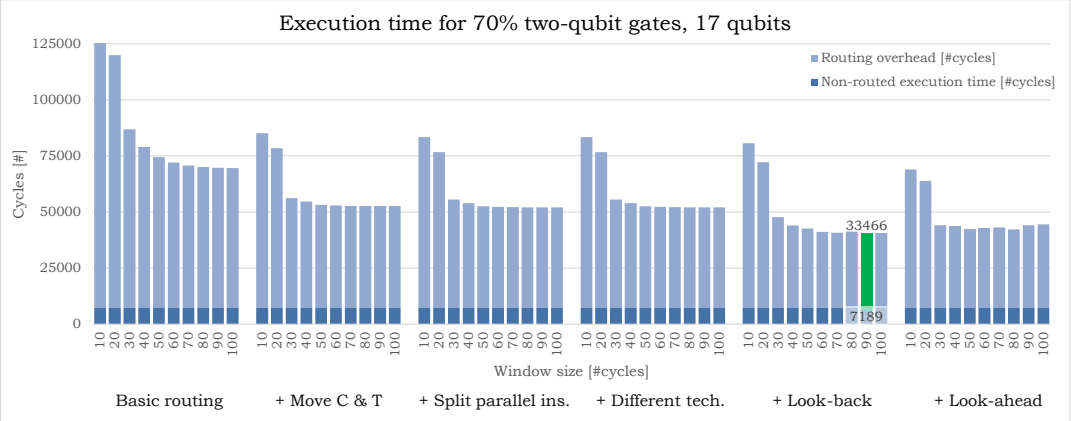


Figure A.24: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 70% two-qubit gates for 17 qubits.

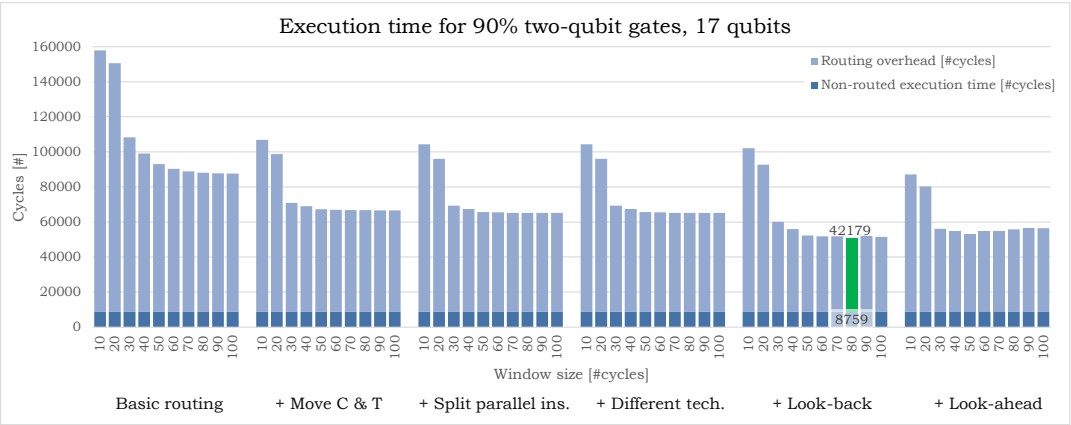


Figure A.25: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 90% two-qubit gates for 17 qubits.

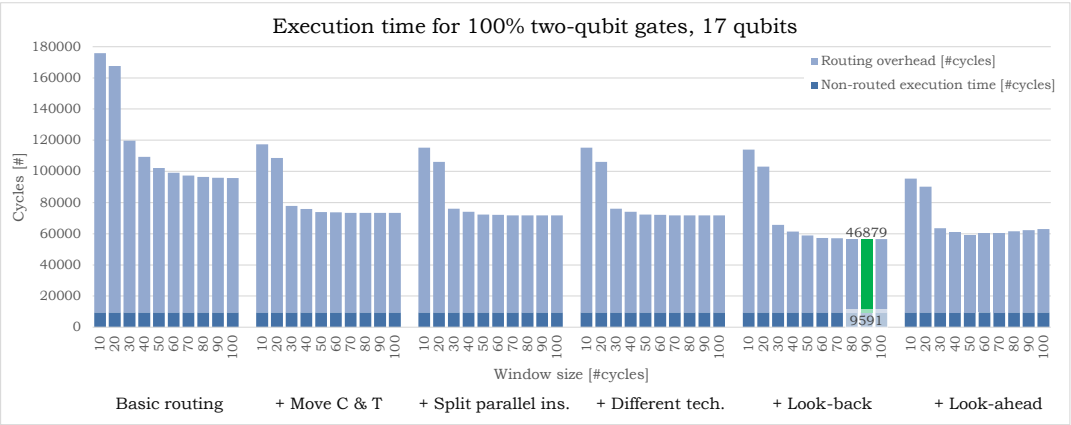


Figure A.26: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 100% two-qubit gates for 17 qubits.

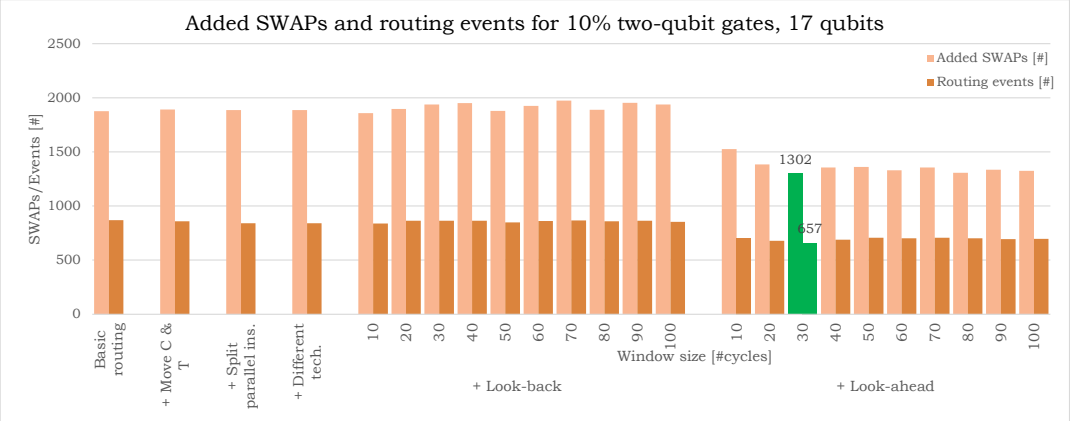


Figure A.27: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 10% two-qubit gates for 17 qubits.

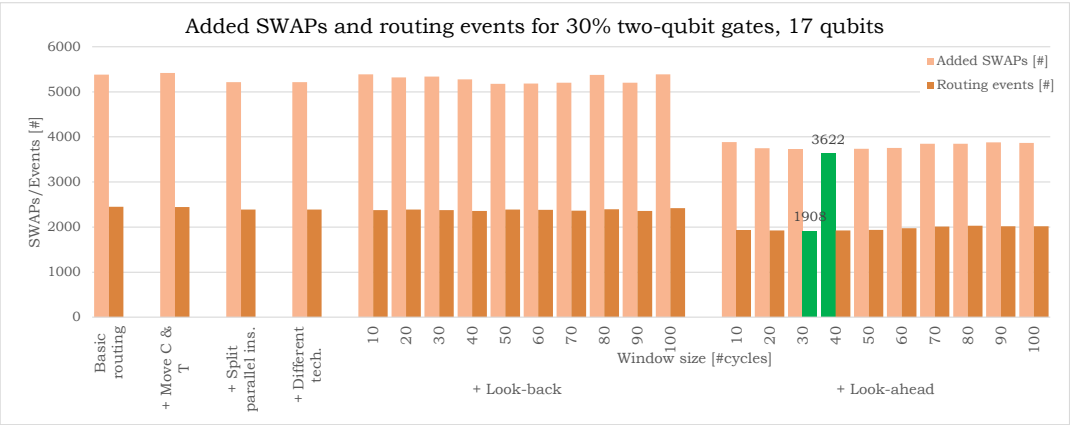


Figure A.28: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 30% two-qubit gates for 17 qubits.

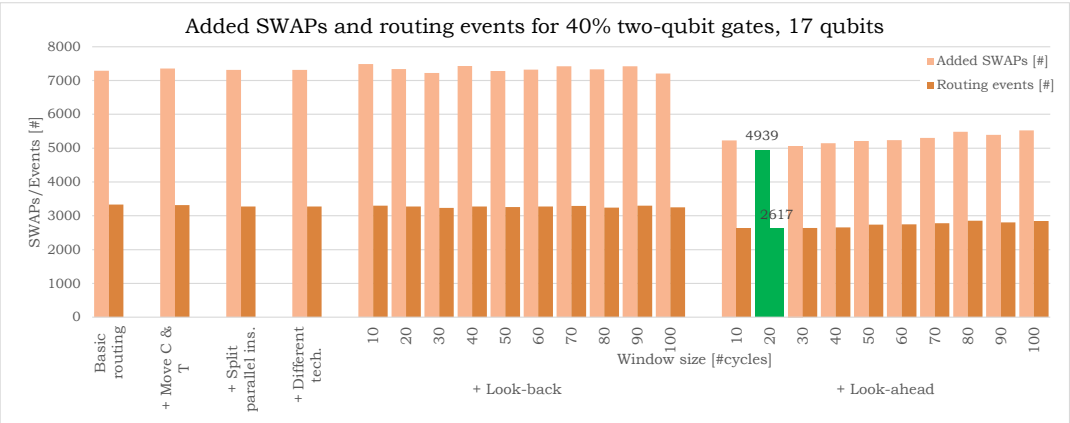


Figure A.29: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 40% two-qubit gates for 17 qubits.

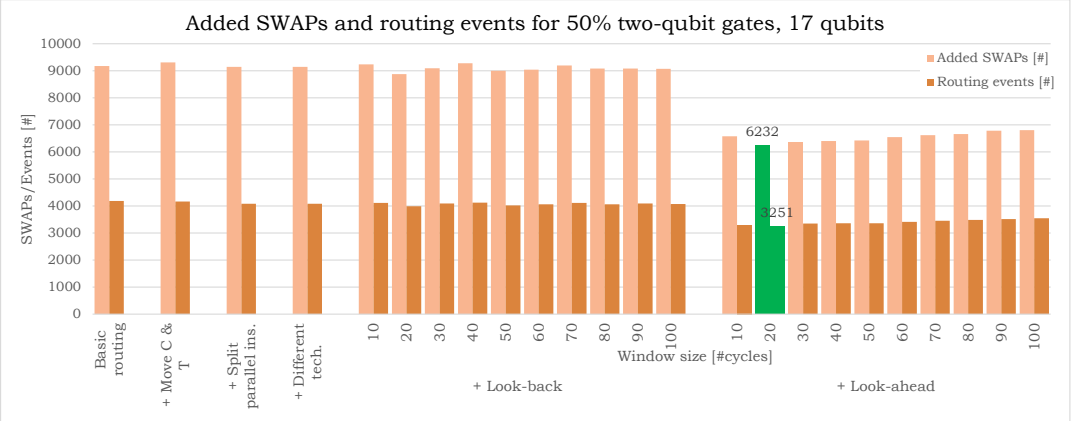


Figure A.30: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 50% two-qubit gates for 17 qubits.

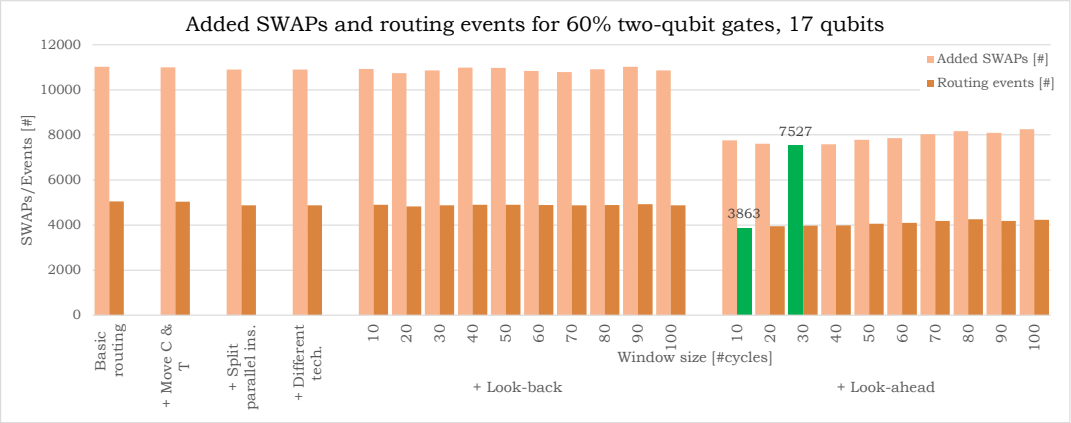


Figure A.31: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 60% two-qubit gates for 17 qubits.

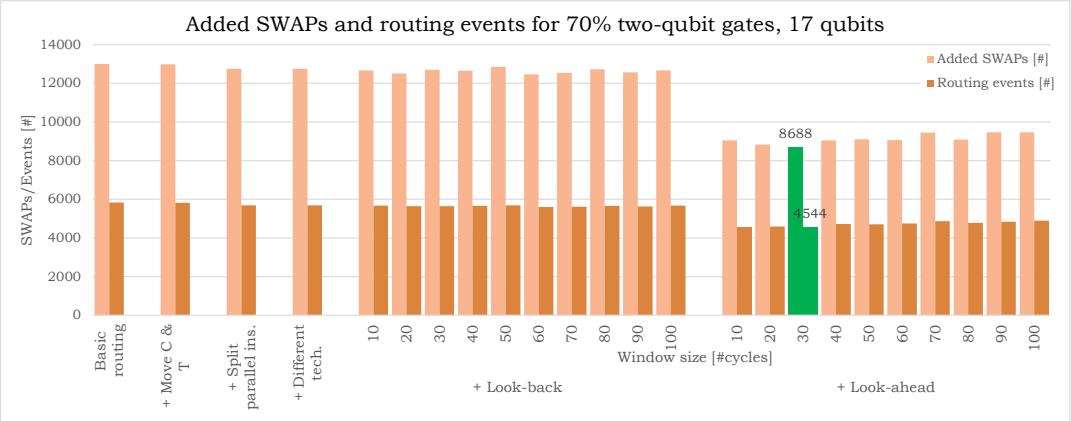


Figure A.32: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 170% two-qubit gates for 17 qubits.

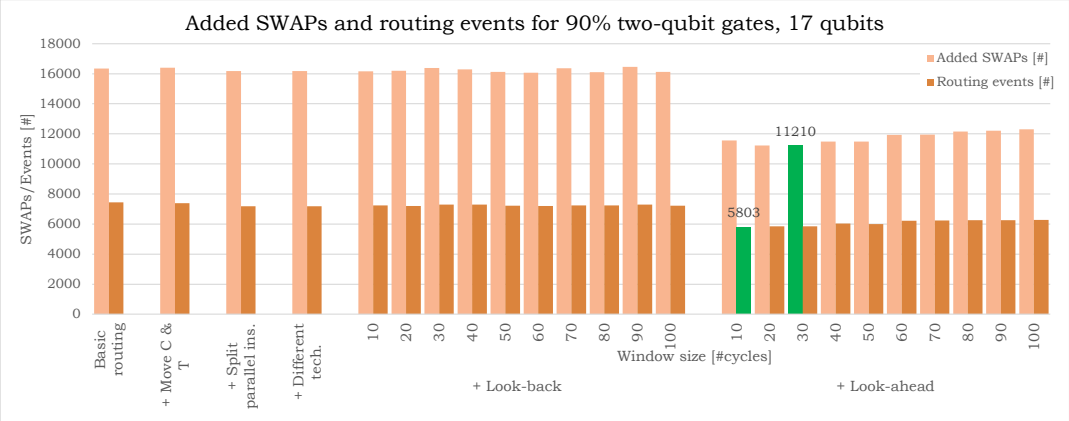


Figure A.33: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 90% two-qubit gates for 17 qubits.

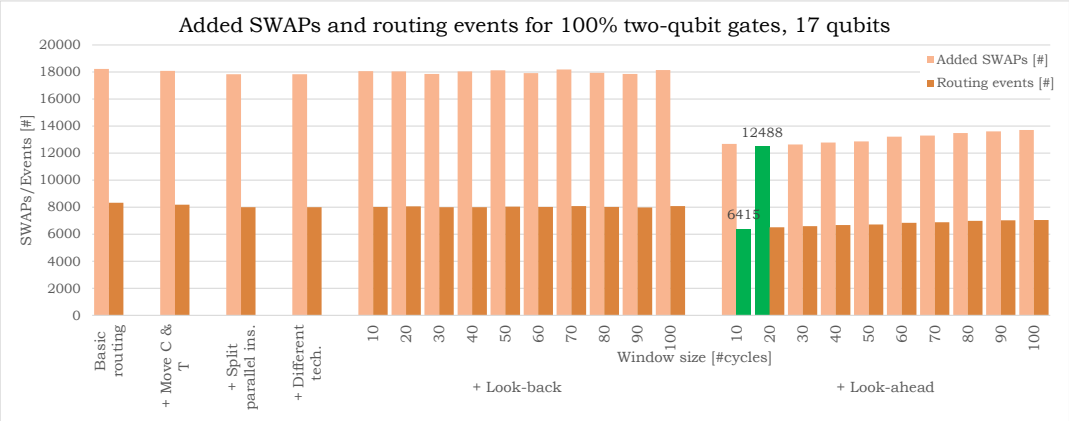


Figure A.34: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 100% two-qubit gates for 17 qubits.

A.3. 49 qubits

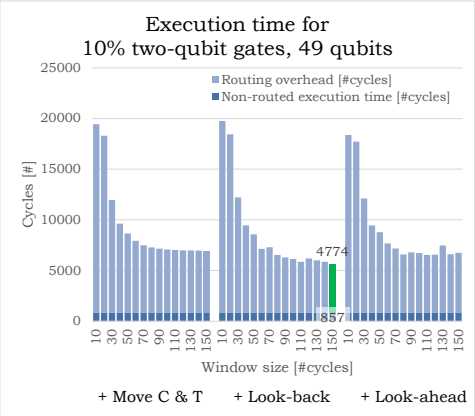


Figure A.35: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 10% two-qubit gates for 49 qubits.

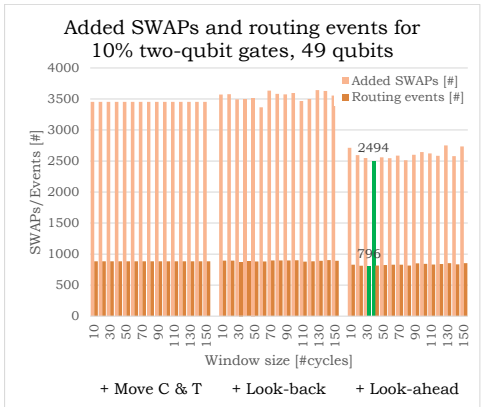


Figure A.36: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 10% two-qubit gates for 49 qubits.

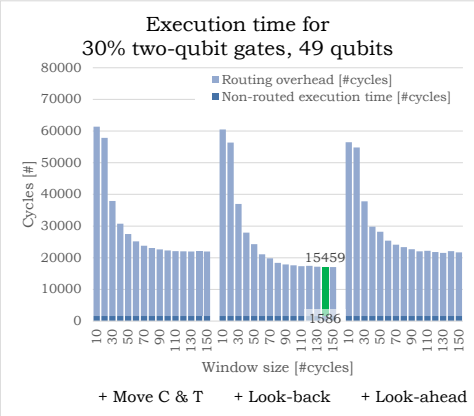


Figure A.37: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 30% two-qubit gates for 49 qubits.

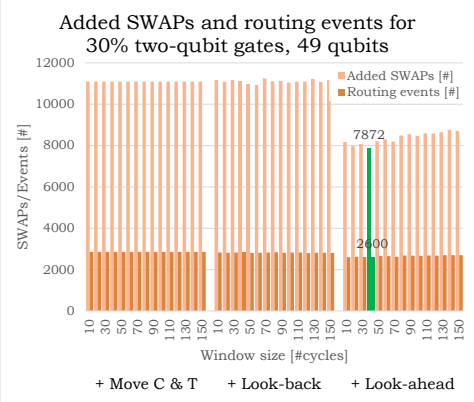


Figure A.38: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 30% two-qubit gates for 49 qubits.

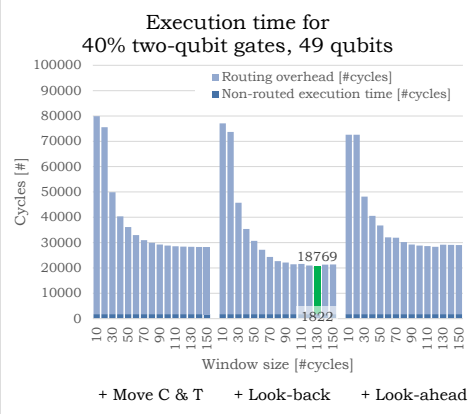


Figure A.39: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 40% two-qubit gates for 49 qubits.

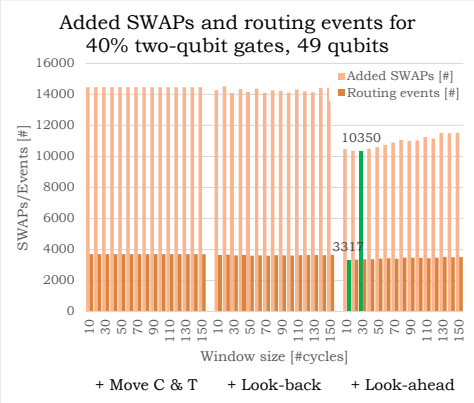


Figure A.40: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 40% two-qubit gates for 49 qubits.

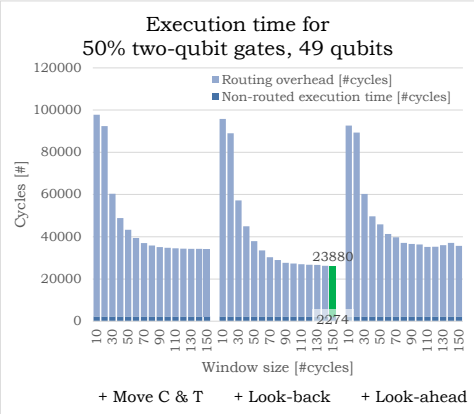


Figure A.41: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 50% two-qubit gates for 49 qubits.

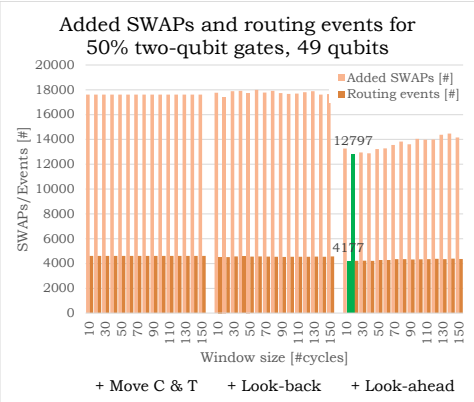


Figure A.42: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 50% two-qubit gates for 49 qubits.

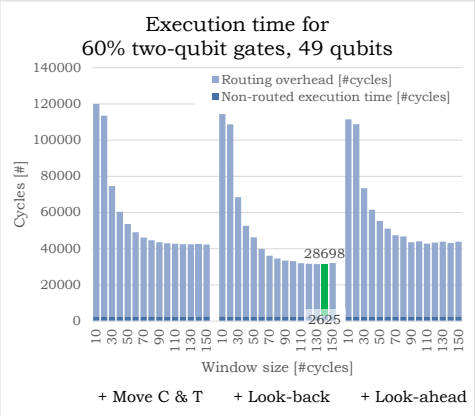


Figure A.43: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 60% two-qubit gates for 49 qubits.

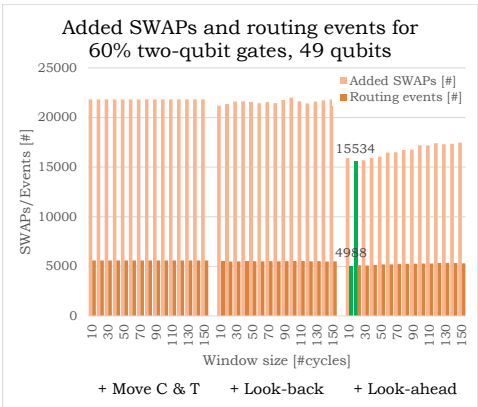


Figure A.44: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 60% two-qubit gates for 49 qubits.

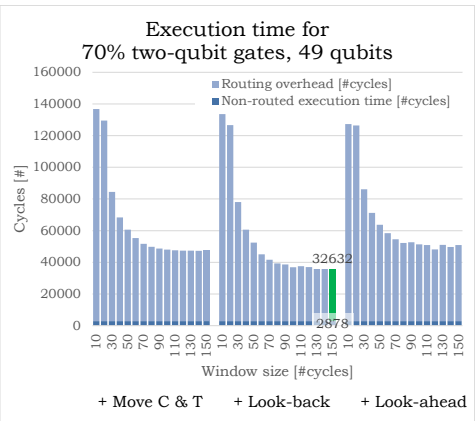


Figure A.45: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 70% two-qubit gates for 49 qubits.

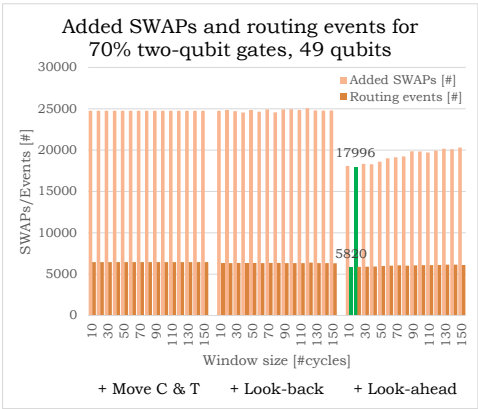


Figure A.46: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 70% two-qubit gates for 49 qubits.

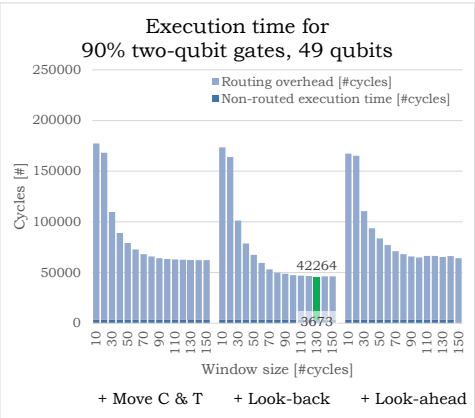


Figure A.47: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 90% two-qubit gates for 49 qubits.

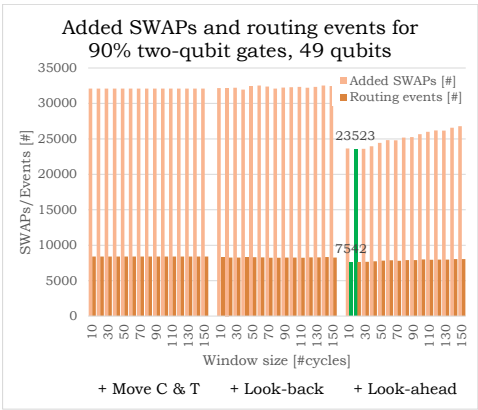


Figure A.48: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 90% two-qubit gates for 49 qubits.

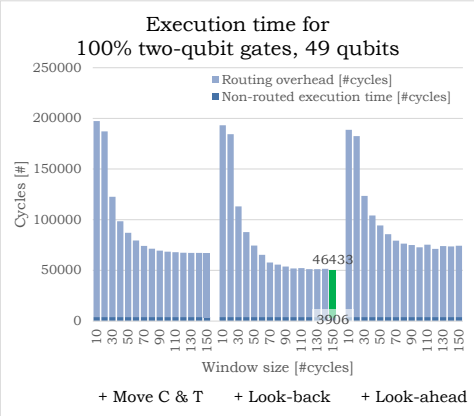


Figure A.49: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains 90% two-qubit gates for 49 qubits.

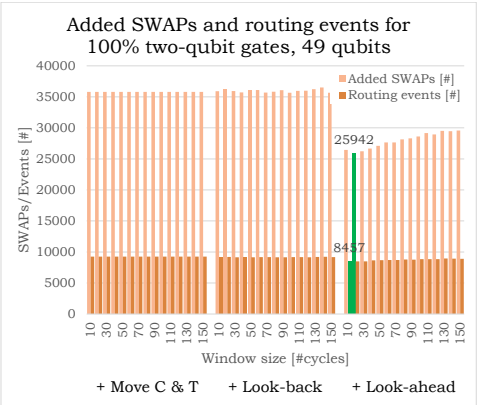


Figure A.50: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains 100% two-qubit gates for 49 qubits.

B

Routing results for different extra path lengths

B.1. 7 qubit

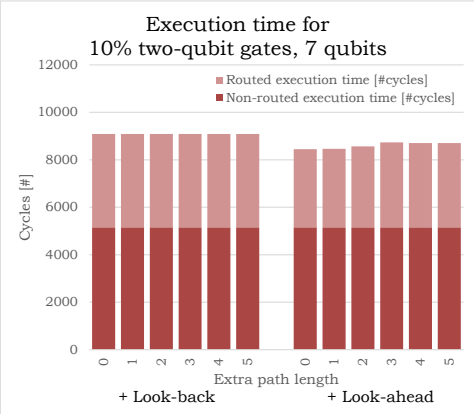


Figure B.1: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 10% two-qubit gates for 7 qubits.

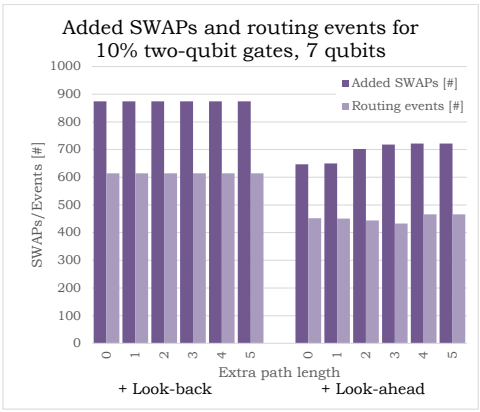


Figure B.2: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 10% two-qubit gates for 7 qubits.

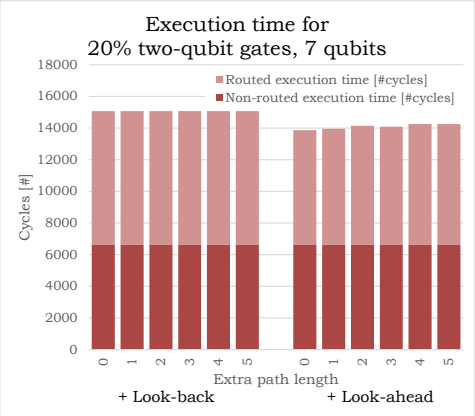


Figure B.3: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 20% two-qubit gates for 7 qubits.

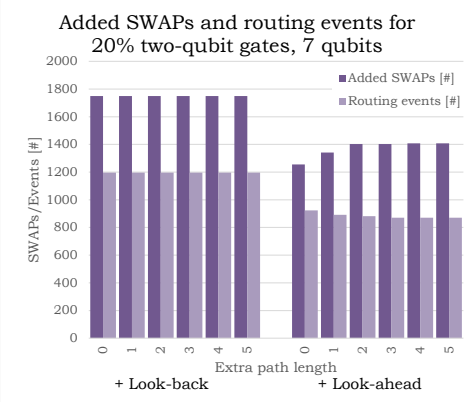


Figure B.4: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 20% two-qubit gates for 7 qubits.

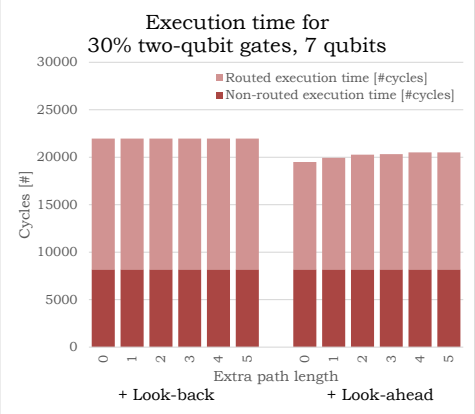


Figure B.5: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 30% two-qubit gates for 7 qubits.

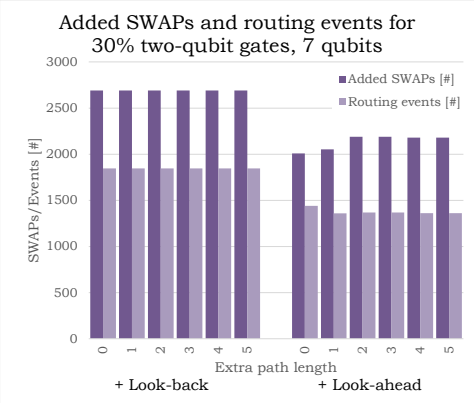


Figure B.6: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 30% two-qubit gates for 7 qubits.

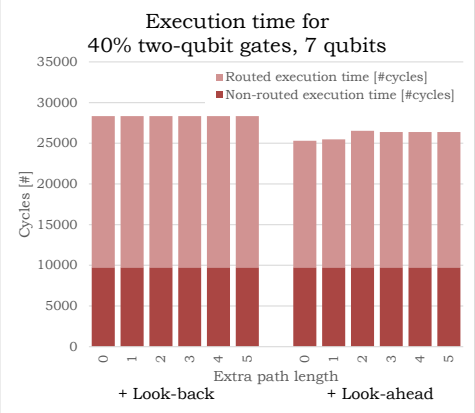


Figure B.7: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 40% two-qubit gates for 7 qubits.

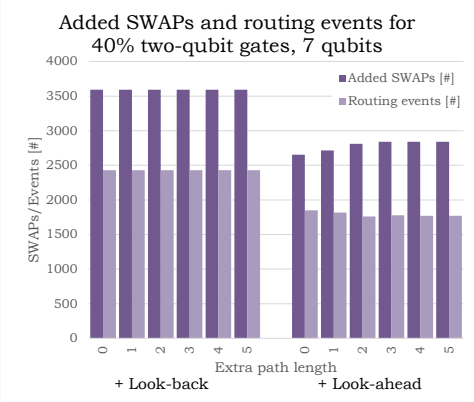


Figure B.8: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 40% two-qubit gates for 7 qubits.

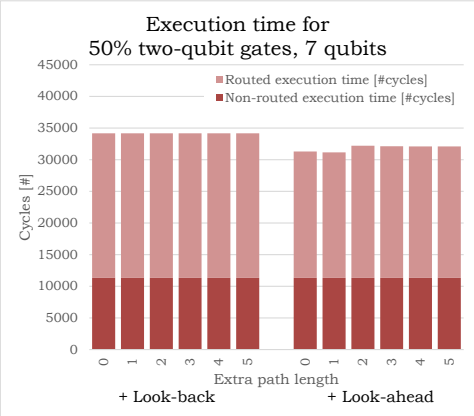


Figure B.9: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 50% two-qubit gates for 7 qubits.

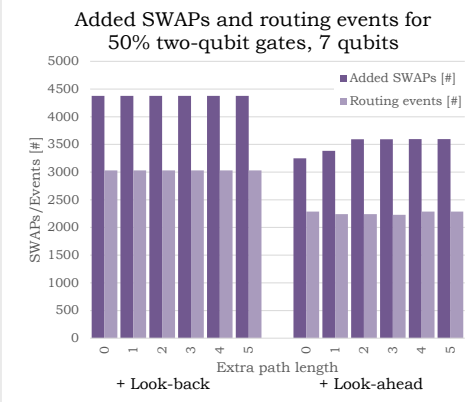


Figure B.10: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 50% two-qubit gates for 7 qubits.

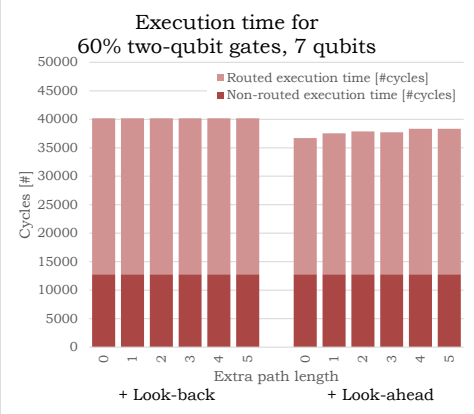


Figure B.11: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 60% two-qubit gates for 7 qubits.

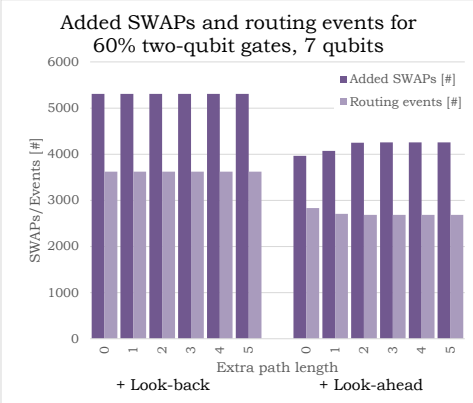


Figure B.12: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 60% two-qubit gates for 7 qubits.

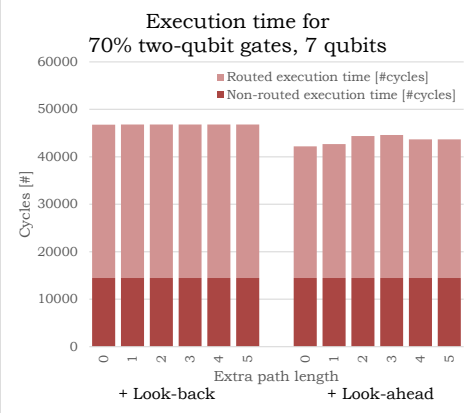


Figure B.13: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 70% two-qubit gates for 7 qubits.

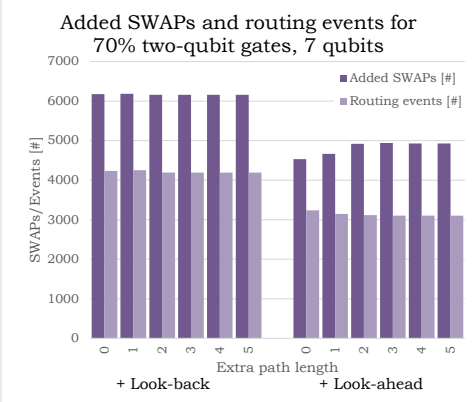


Figure B.14: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 70% two-qubit gates for 7 qubits.

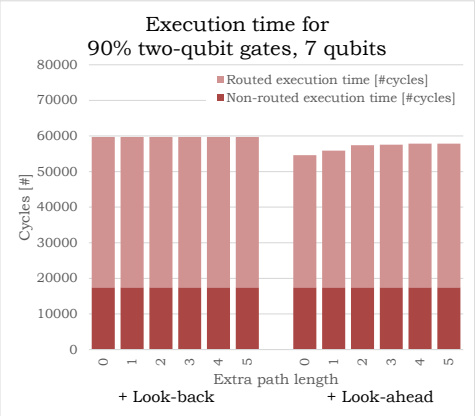


Figure B.15: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 90% two-qubit gates for 7 qubits.

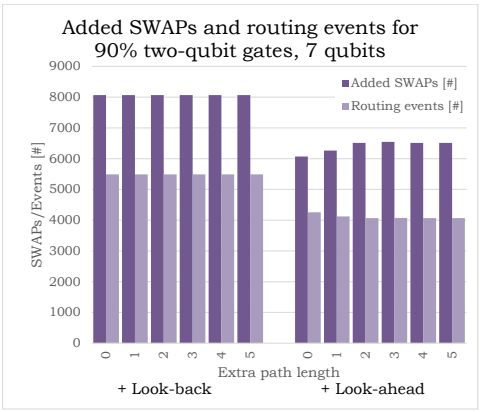


Figure B.16: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 90% two-qubit gates for 7 qubits.

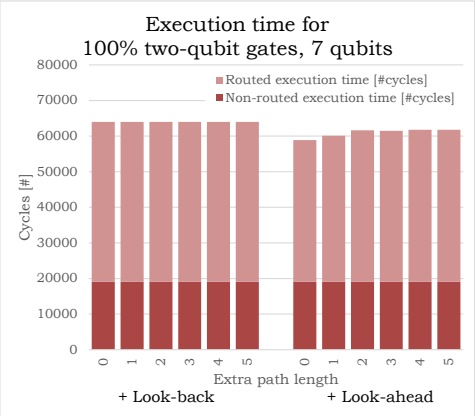


Figure B.17: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 90% two-qubit gates for 7 qubits.

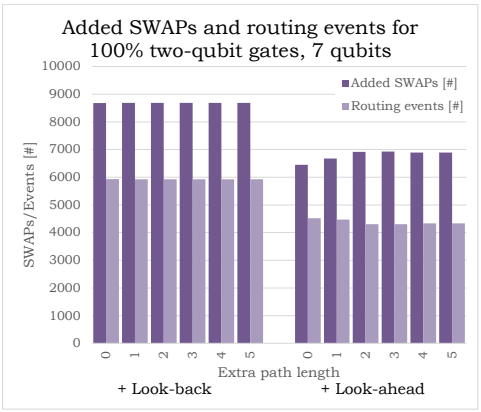


Figure B.18: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 100% two-qubit gates for 7 qubits.

B.2. 17 qubit

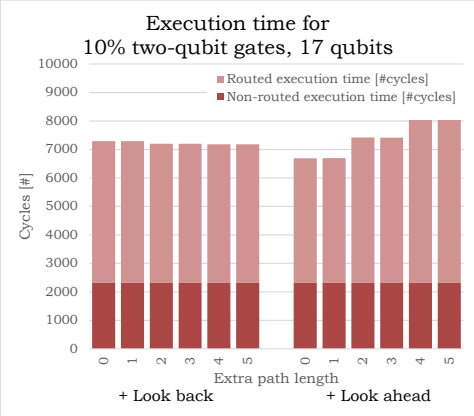


Figure B.19: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 10% two-qubit gates for 17 qubits.

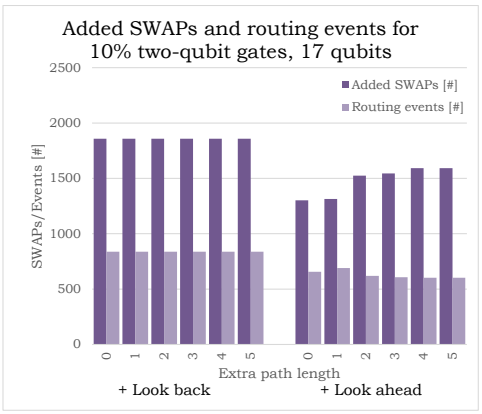


Figure B.20: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 10% two-qubit gates for 17 qubits.

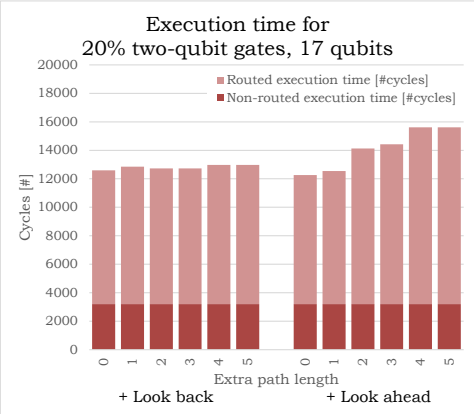


Figure B.21: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 20% two-qubit gates for 17 qubits.

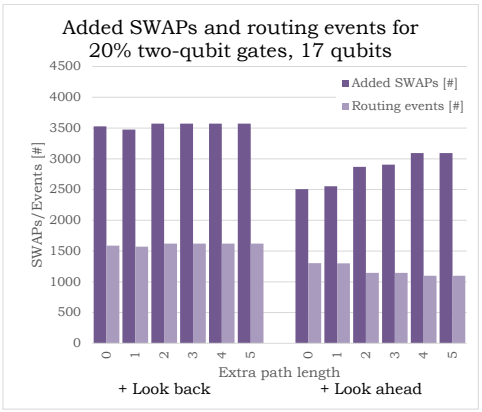


Figure B.22: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 20% two-qubit gates for 17 qubits.

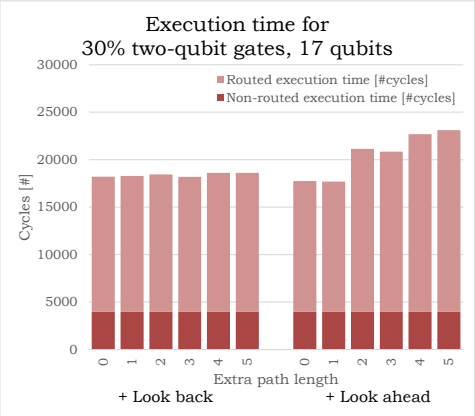


Figure B.23: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 30% two-qubit gates for 17 qubits.

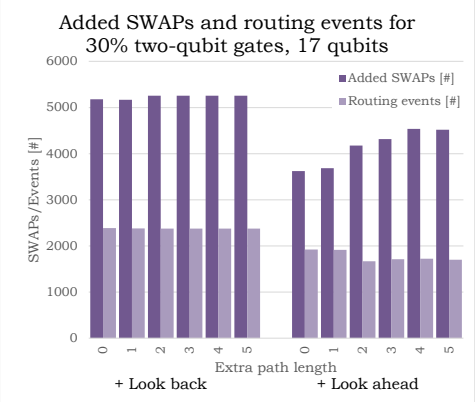


Figure B.24: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 30% two-qubit gates for 17 qubits.

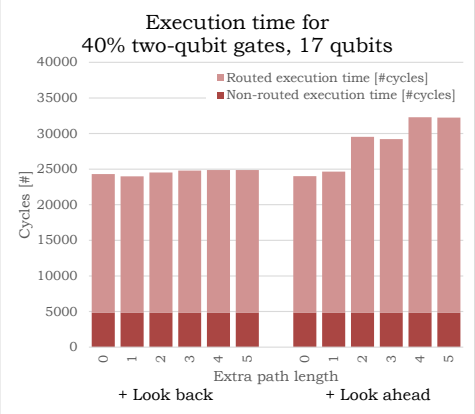


Figure B.25: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 40% two-qubit gates for 17 qubits.

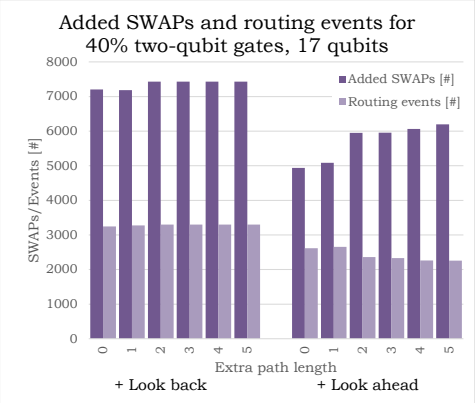


Figure B.26: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 40% two-qubit gates for 17 qubits.

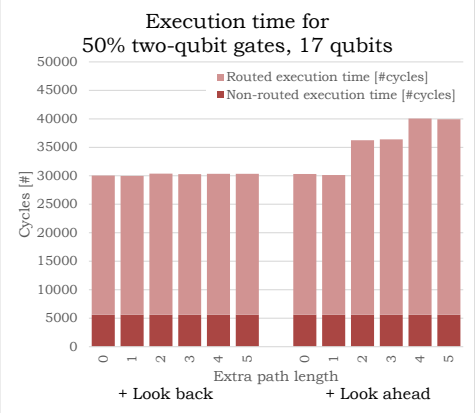


Figure B.27: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 50% two-qubit gates for 17 qubits.

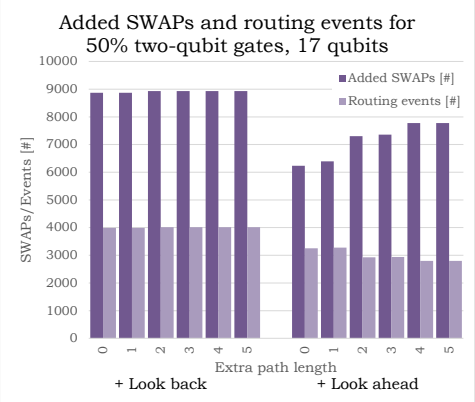


Figure B.28: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 50% two-qubit gates for 17 qubits.

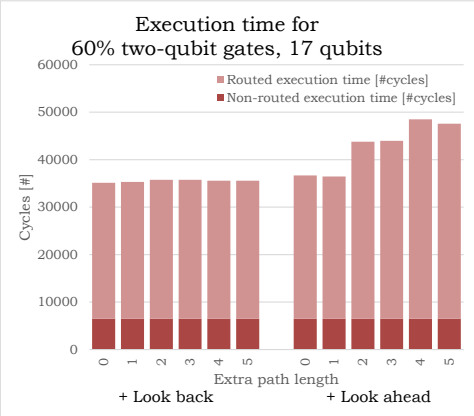


Figure B.29: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 60% two-qubit gates for 17 qubits.

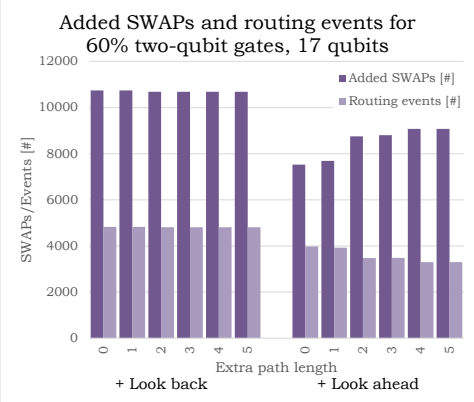


Figure B.30: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 60% two-qubit gates for 17 qubits.

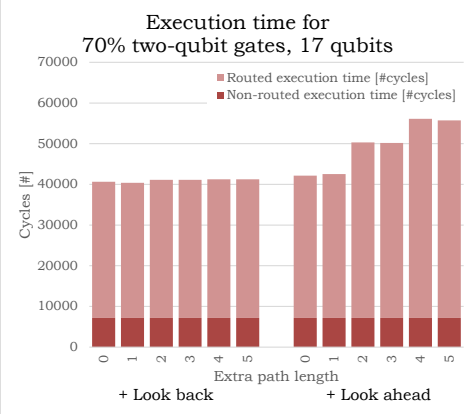


Figure B.31: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 70% two-qubit gates for 17 qubits.

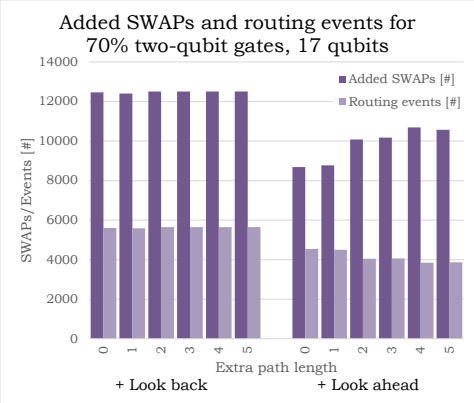


Figure B.32: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 70% two-qubit gates for 17 qubits.

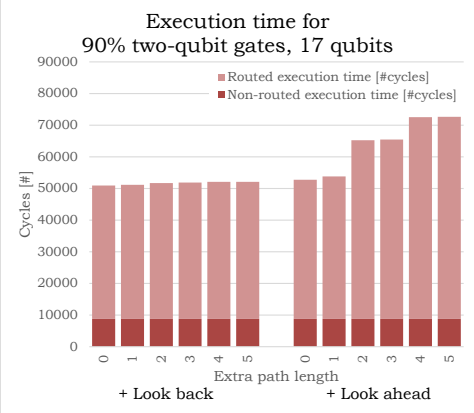


Figure B.33: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 90% two-qubit gates for 17 qubits.

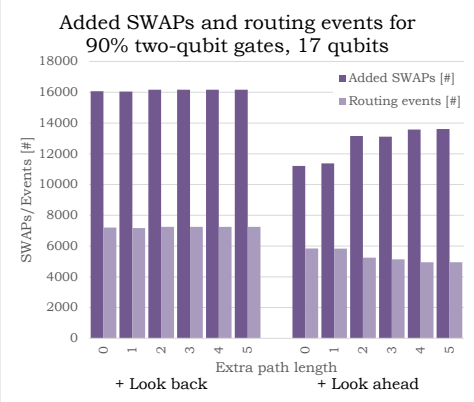


Figure B.34: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 90% two-qubit gates for 17 qubits.

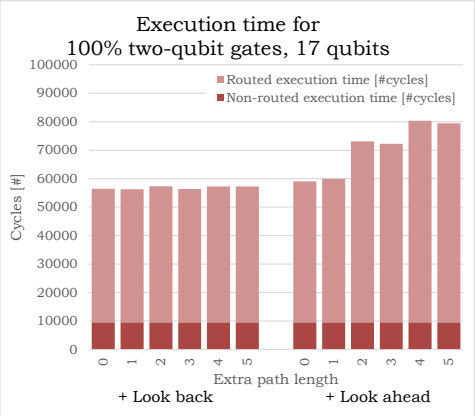


Figure B.35: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 90% two-qubit gates for 17 qubits.

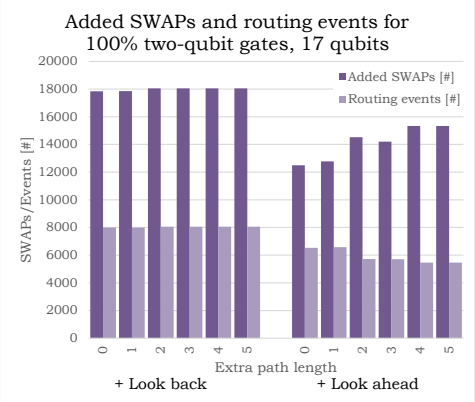


Figure B.36: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 100% two-qubit gates for 17 qubits.

B.3. 49 qubits

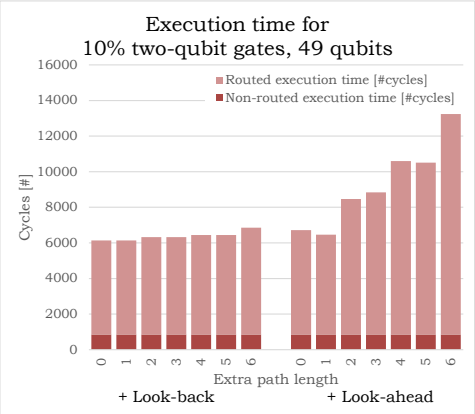


Figure B.37: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 10% two-qubit gates for 49 qubits.

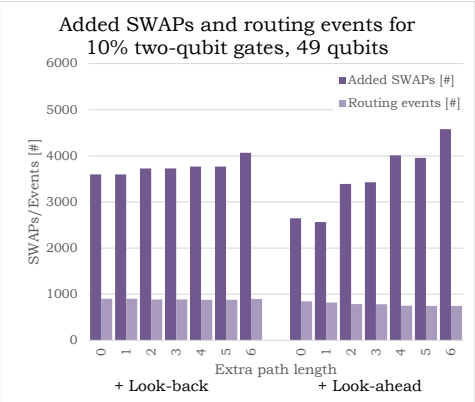


Figure B.38: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 10% two-qubit gates for 49 qubits.

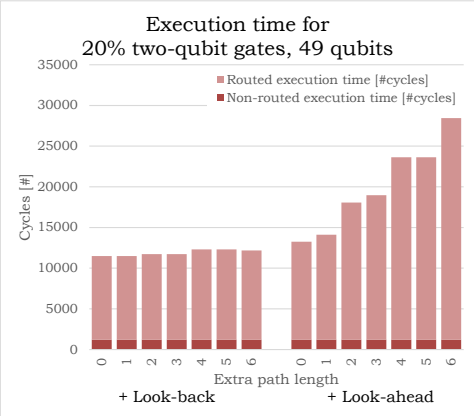


Figure B.39: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 20% two-qubit gates for 49 qubits.

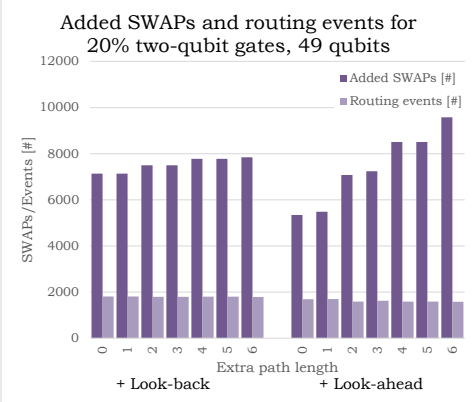


Figure B.40: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 20% two-qubit gates for 49 qubits.

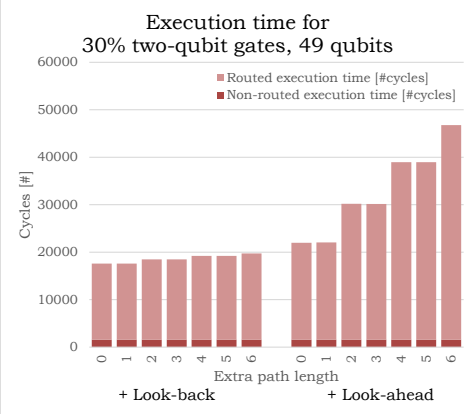


Figure B.41: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 30% two-qubit gates for 49 qubits.

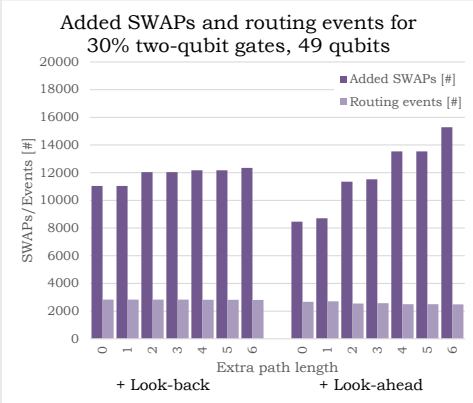


Figure B.42: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 30% two-qubit gates for 49 qubits.

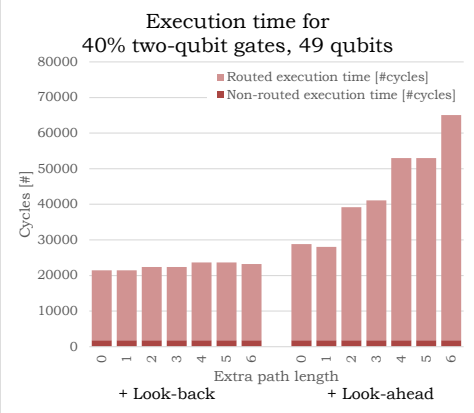


Figure B.43: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 40% two-qubit gates for 49 qubits.

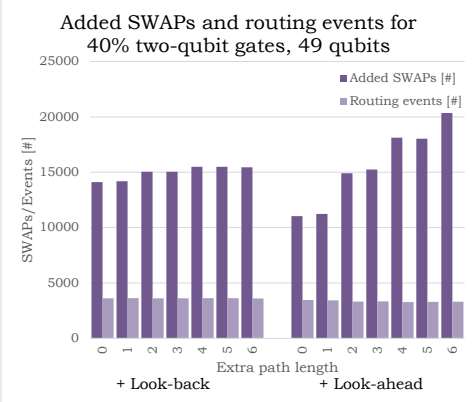


Figure B.44: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 40% two-qubit gates for 49 qubits.

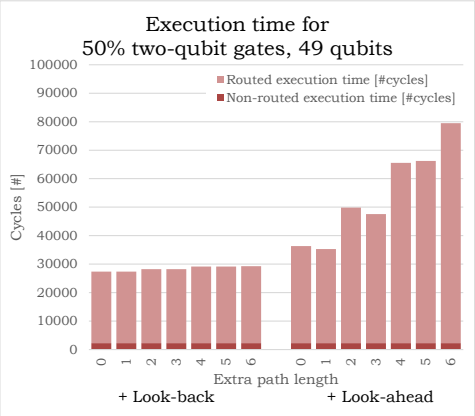


Figure B.45: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 50% two-qubit gates for 49 qubits.

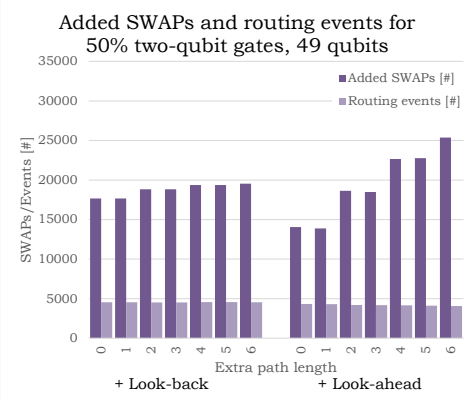


Figure B.46: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 50% two-qubit gates for 49 qubits.

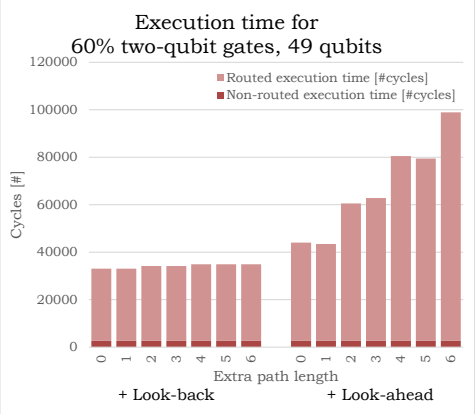


Figure B.47: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 60% two-qubit gates for 49 qubits.

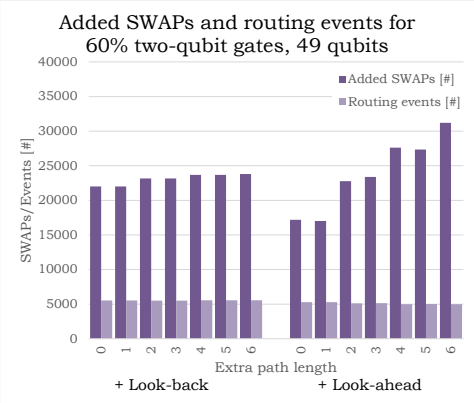


Figure B.48: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 60% two-qubit gates for 49 qubits.

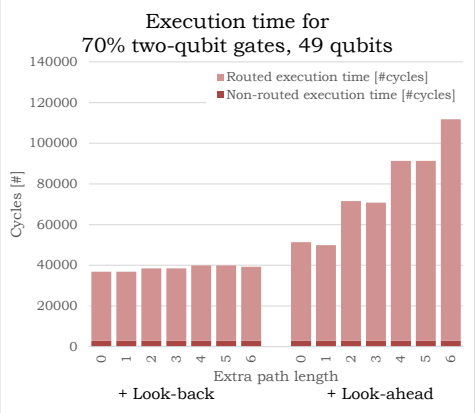


Figure B.49: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 70% two-qubit gates for 49 qubits.

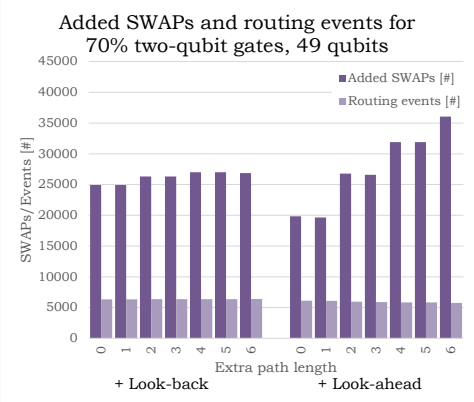


Figure B.50: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 70% two-qubit gates for 49 qubits.

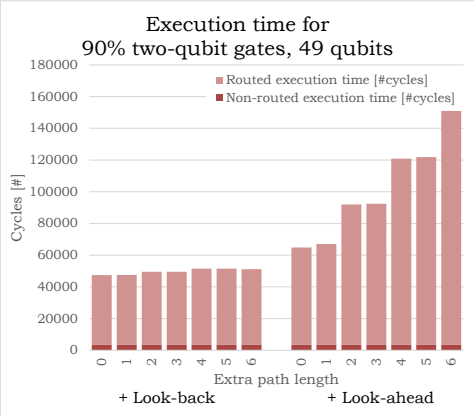


Figure B.51: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 90% two-qubit gates for 49 qubits.

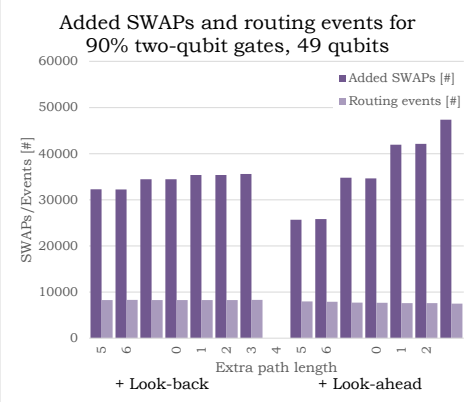


Figure B.52: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 90% two-qubit gates for 49 qubits.

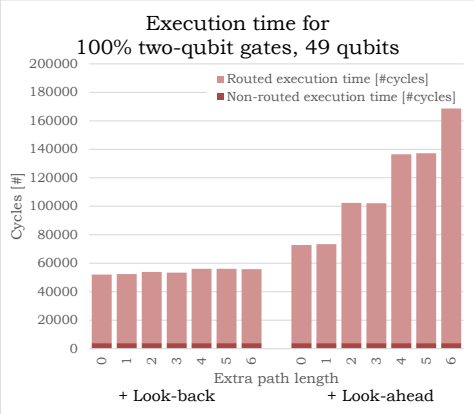


Figure B.53: Quantum circuit execution time in cycles for different extra path lengths to allow detours. The random benchmark contains 90% two-qubit gates for 49 qubits.

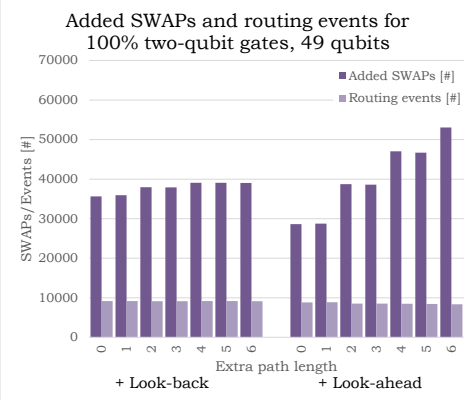


Figure B.54: Number of routing events and added SWAPs for different extra path lengths to allow detours. The random benchmark contains 100% two-qubit gates for 49 qubits.

C

Routing results for different single-qubit gate durations

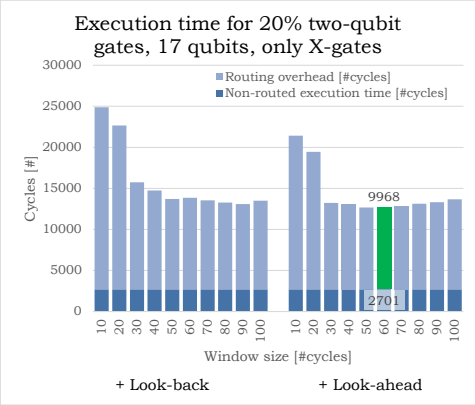


Figure C.1: Quantum circuit execution time in cycles. The random benchmark contains 80% X-gates and 20% two-qubit gates for 17 qubits.

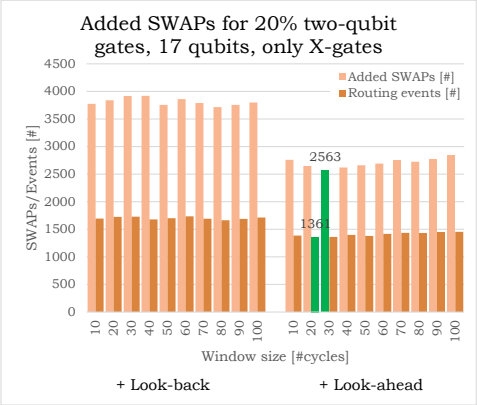


Figure C.2: Number of routing events and added SWAPs. The random benchmark contains 80% X-gates and 20% two-qubit gates for 17 qubits.

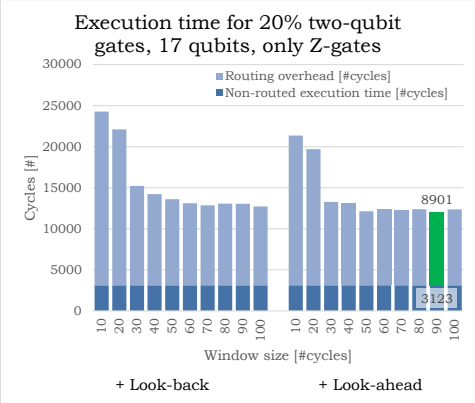


Figure C.3: Quantum circuit execution time in cycles. The random benchmark contains 80% Z-gates and 20% two-qubit gates for 17 qubits.

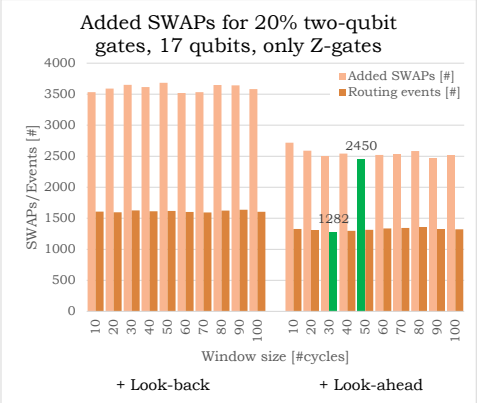


Figure C.4: Number of routing events and added SWAPs. The random benchmark contains 80% Z-gates and 20% two-qubit gates for 17 qubits.

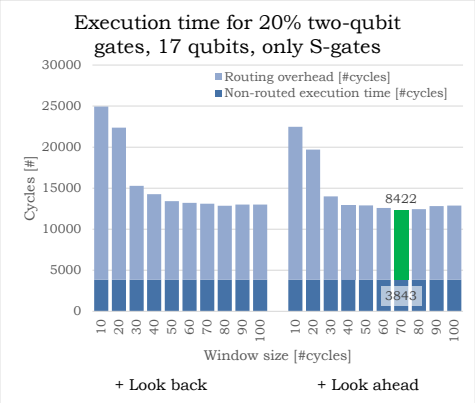


Figure C.5: Quantum circuit execution time in cycles. The random benchmark contains 80% S-gates and 20% two-qubit gates for 17 qubits.

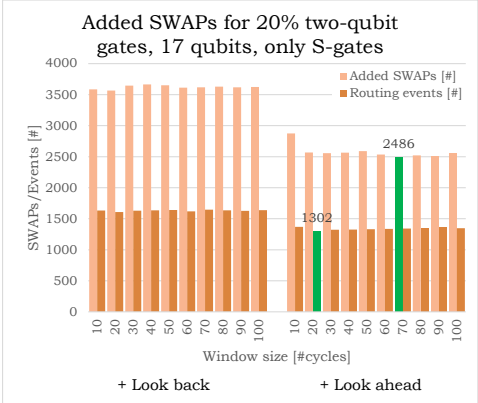


Figure C.6: Number of routing events and added SWAPs. The random benchmark contains 80% S-gates and 20% two-qubit gates for 17 qubits.

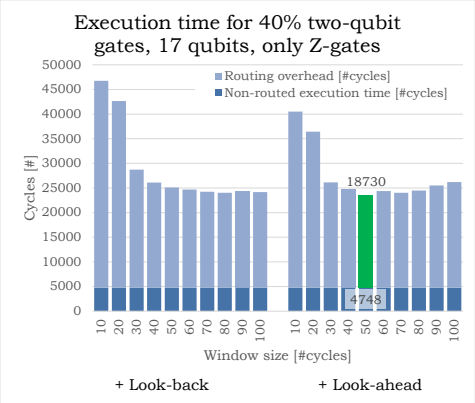


Figure C.7: Quantum circuit execution time in cycles. The random benchmark contains 60% Z-gates and 40% two-qubit gates for 17 qubits.

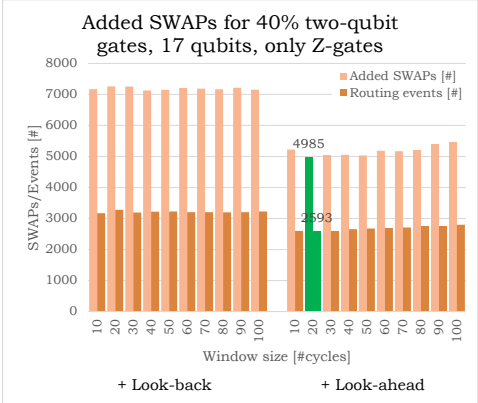


Figure C.8: Number of routing events and added SWAPs. The random benchmark contains 60% Z-gates and 40% two-qubit gates for 17 qubits.

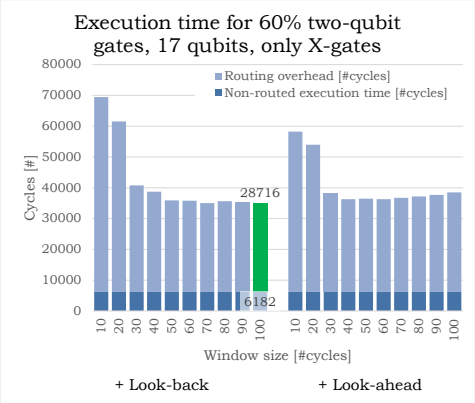


Figure C.9: Quantum circuit execution time in cycles. The random benchmark contains 40% X-gates and 60% two-qubit gates for 17 qubits.

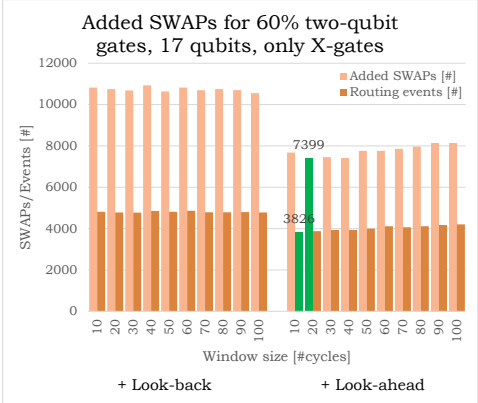


Figure C.10: Number of routing events and added SWAPs. The random benchmark contains 40% X-gates and 60% two-qubit gates for 17 qubits.

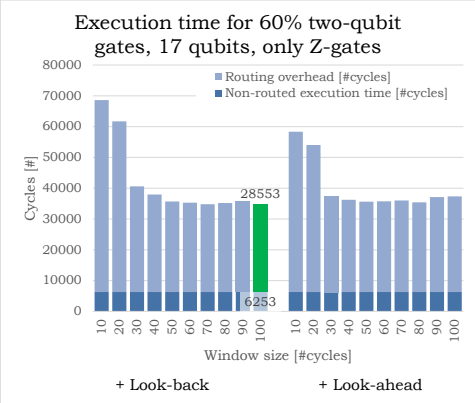


Figure C.11: Quantum circuit execution time in cycles. The random benchmark contains 40% Z-gates and 60% two-qubit gates for 17 qubits.

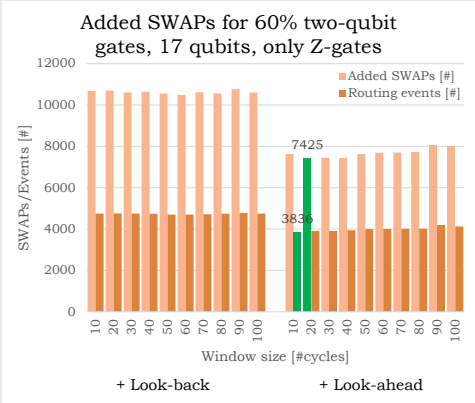


Figure C.12: Number of routing events and added SWAPs. The random benchmark contains 40% Z-gates and 60% two-qubit gates for 17 qubits.

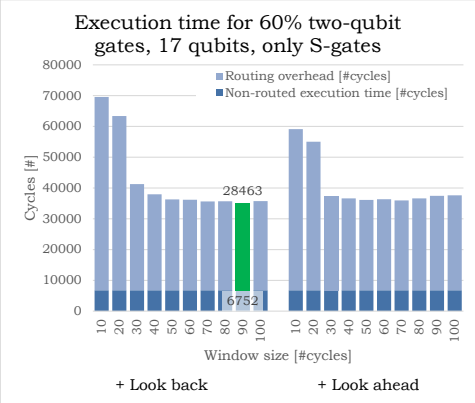


Figure C.13: Quantum circuit execution time in cycles. The random benchmark contains 40% S-gates and 60% two-qubit gates for 17 qubits.

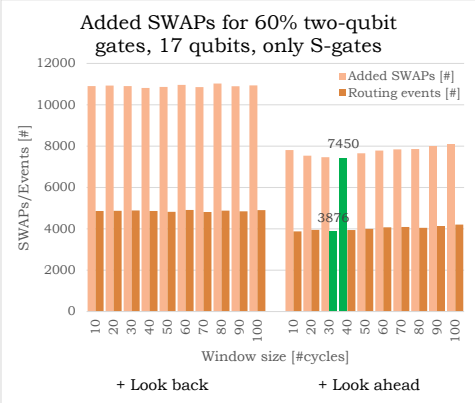


Figure C.14: Number of routing events and added SWAPs. The random benchmark contains 40% S-gates and 60% two-qubit gates for 17 qubits.

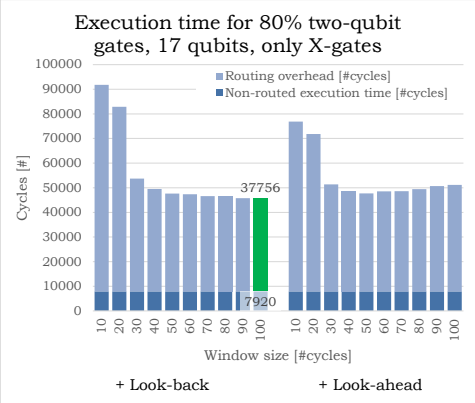


Figure C.15: Quantum circuit execution time in cycles. The random benchmark contains 20% X-gates and 80% two-qubit gates for 17 qubits.

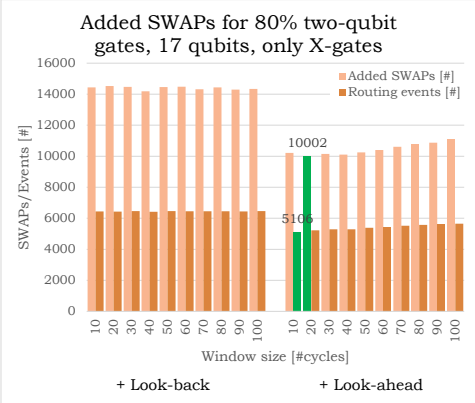


Figure C.16: Number of routing events and added SWAPs. The random benchmark contains 20% X-gates and 80% two-qubit gates for 17 qubits.

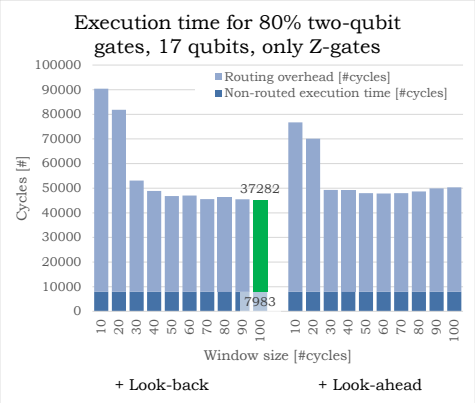


Figure C.17: Quantum circuit execution time in cycles. The random benchmark contains 20% Z-gates and 80% two-qubit gates for 17 qubits.

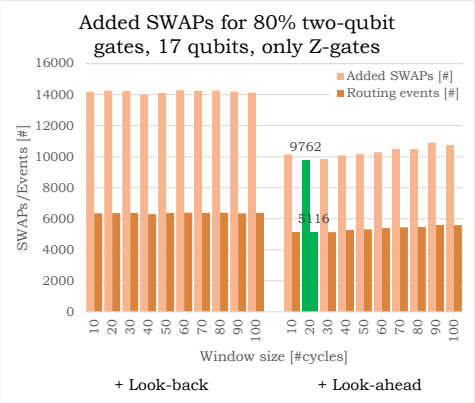


Figure C.18: Number of routing events and added SWAPs. The random benchmark contains 20% Z-gates and 80% two-qubit gates for 17 qubits.

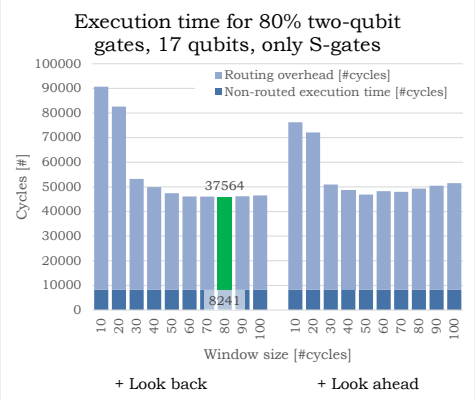


Figure C.19: Quantum circuit execution time in cycles. The random benchmark contains 20% S-gates and 80% two-qubit gates for 17 qubits.

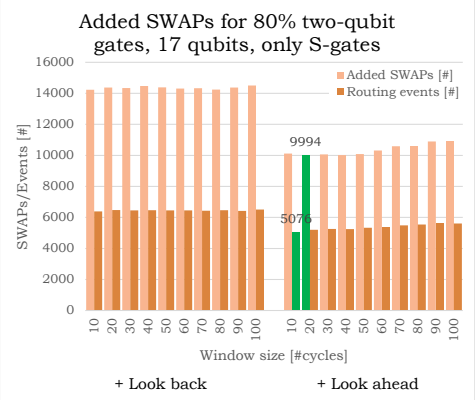


Figure C.20: Number of routing events and added SWAPs. The random benchmark contains 20% S-gates and 80% two-qubit gates for 17 qubits.

D

ScaffCC random benchmarks routing results

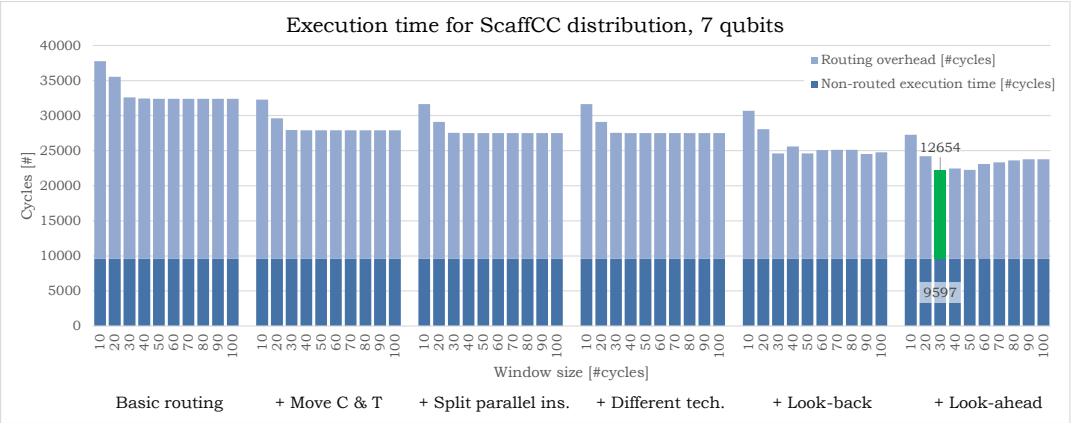


Figure D.1: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains a qubit gate distribution compared to the ScaffCC benchmarks.

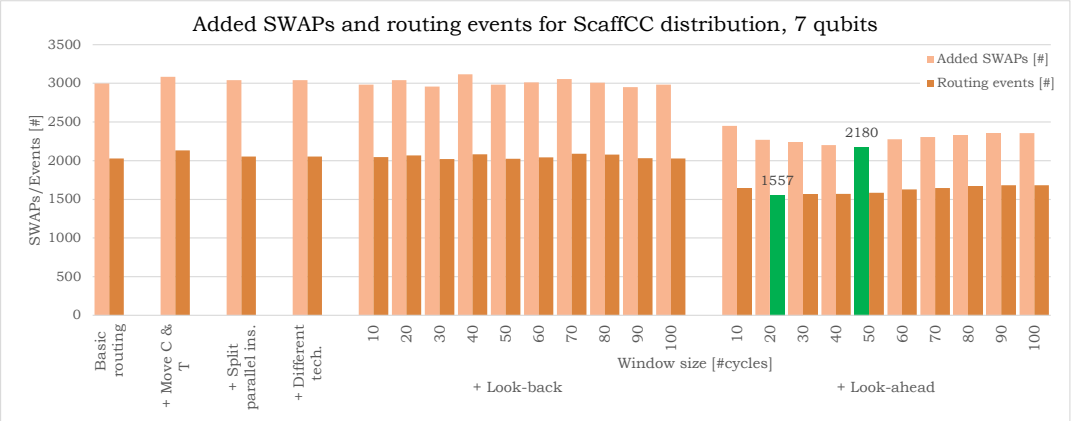


Figure D.2: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains a qubit gate distribution compared to the ScaffCC benchmarks.

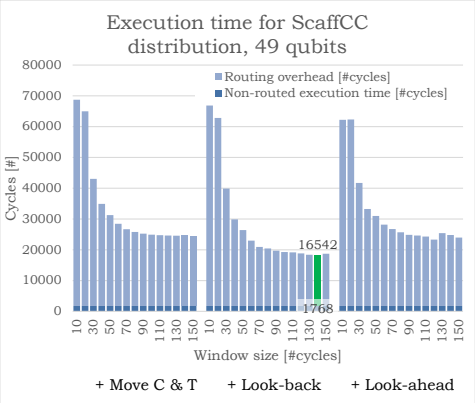


Figure D.3: Quantum circuit execution time in cycles for different functions and window sizes. The random benchmark contains a qubit gate distribution compared to the ScaffCC benchmarks.

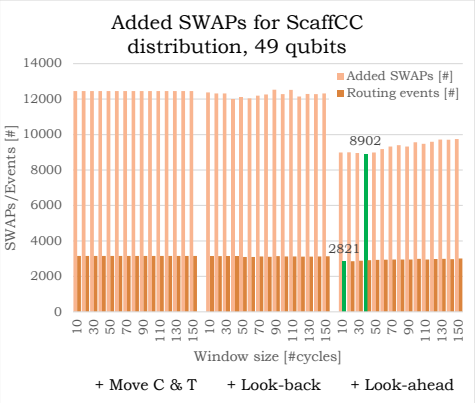


Figure D.4: Number of routing events and added SWAPs for different functions and window sizes. The random benchmark contains a qubit gate distribution compared to the ScaffCC benchmarks.

E

QLib benchmarks routing results

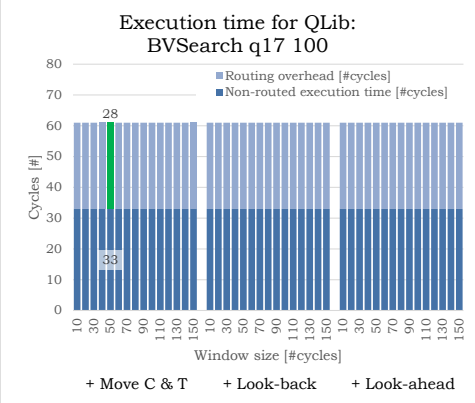


Figure E.1: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: BVSearch 17-qubit number 100 algorithm.

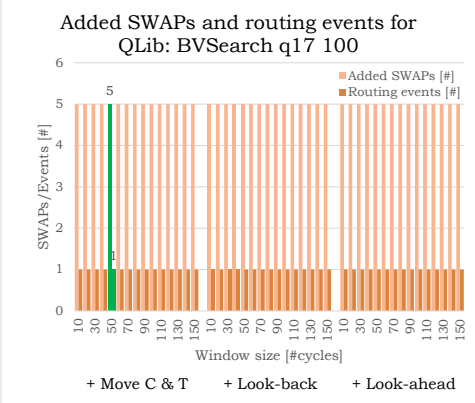


Figure E.2: Number of routing events and added SWAPs for different functions and window size for the QLib: BVSearch 17-qubit number 100 algorithm.

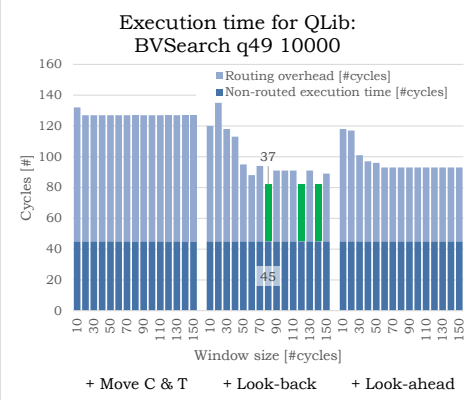


Figure E.3: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: BVSearch 49-qubit number 10000 algorithm.

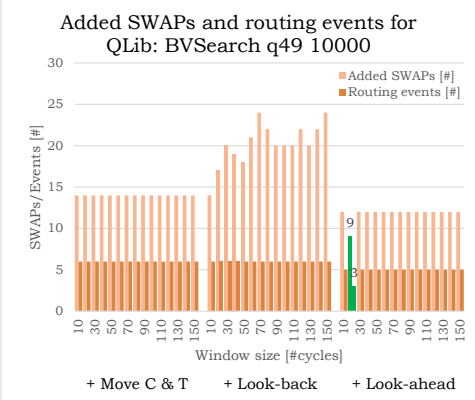


Figure E.4: Number of routing events and added SWAPs for different functions and window size for the QLib: BVSearch 49-qubit number 100000 algorithm.

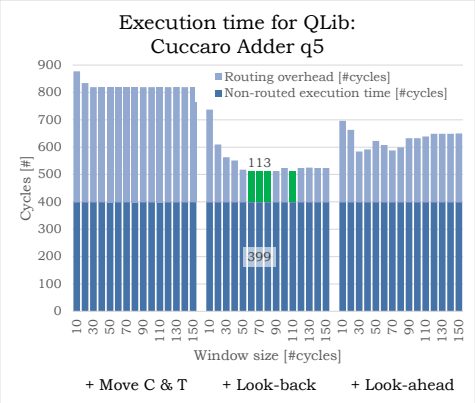


Figure E.5: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: Cuccaro Adder 5-qubit algorithm.

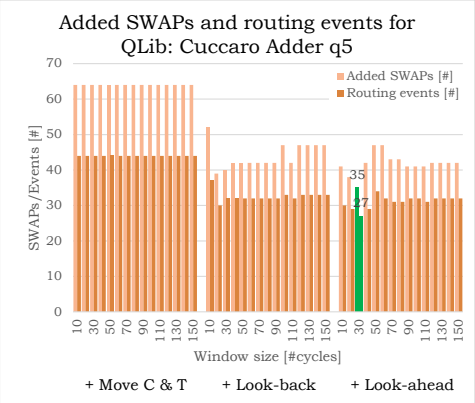


Figure E.6: Number of routing events and added SWAPs for different functions and window size for the QLib: Cuccaro Adder 5-qubit algorithm.

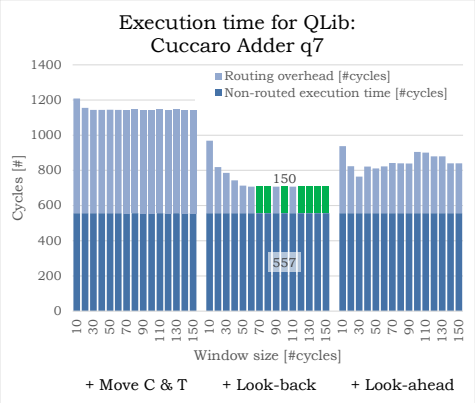


Figure E.7: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: Cuccaro Adder 7-qubit algorithm.

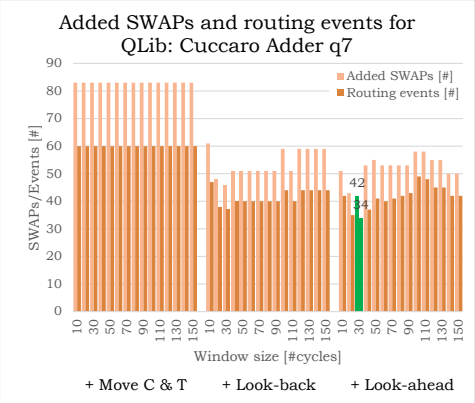


Figure E.8: Number of routing events and added SWAPs for different functions and window size for the QLib: Cuccaro Adder 7-qubit algorithm.

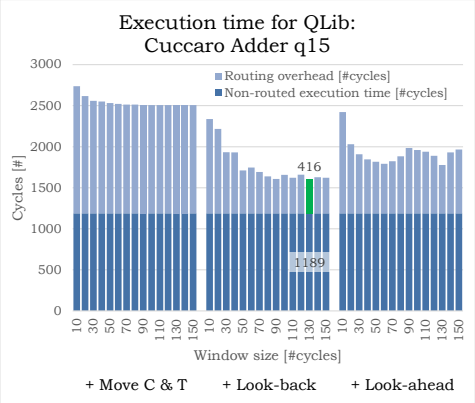


Figure E.9: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: Cuccaro Adder 15-qubit algorithm.

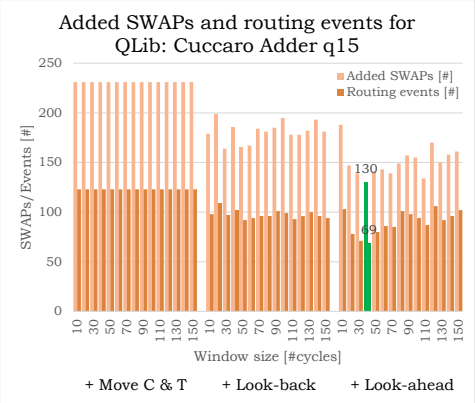


Figure E.10: Number of routing events and added SWAPs for different functions and window size for the QLib: Cuccaro Adder 15-qubit algorithm.

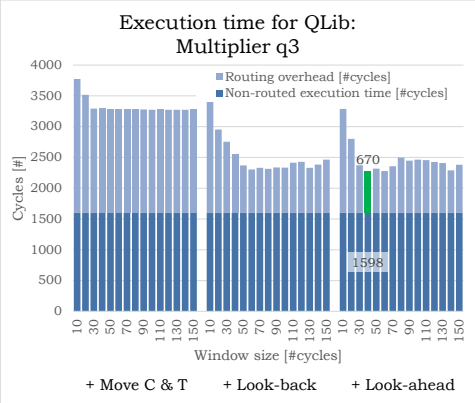


Figure E.11: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: Multiplier 3-qubit algorithm.

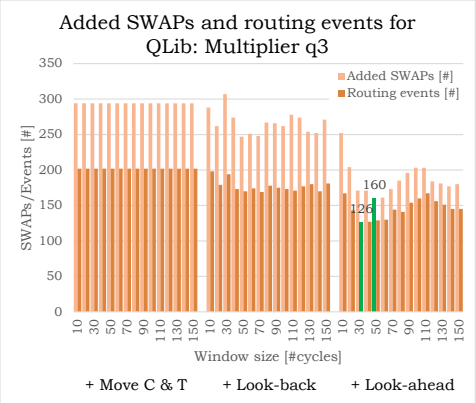


Figure E.12: Number of routing events and added SWAPs for different functions and window size for the QLib: Multiplier 3-qubit algorithm.

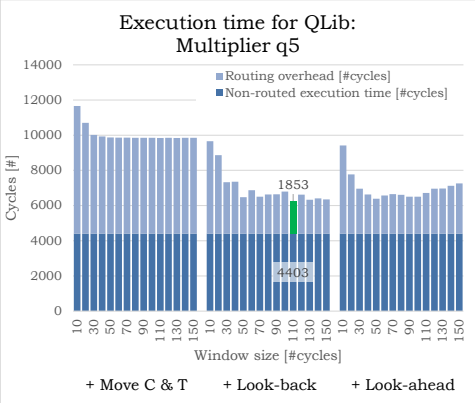


Figure E.13: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: Multiplier 5-qubit algorithm.

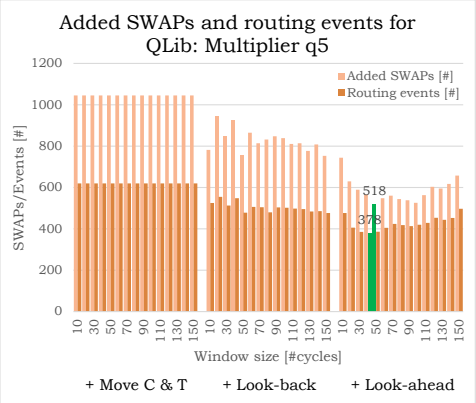


Figure E.14: Number of routing events and added SWAPs for different functions and window size for the QLib: Multiplier 5-qubit algorithm.

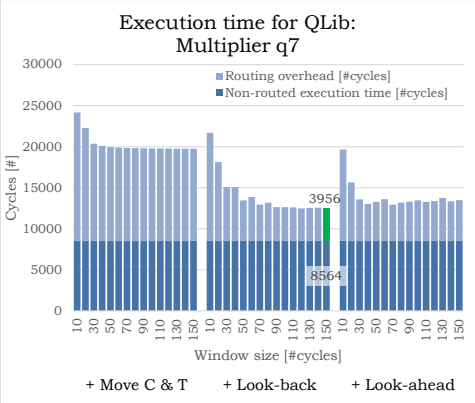


Figure E.15: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: Multiplier 7-qubit algorithm.

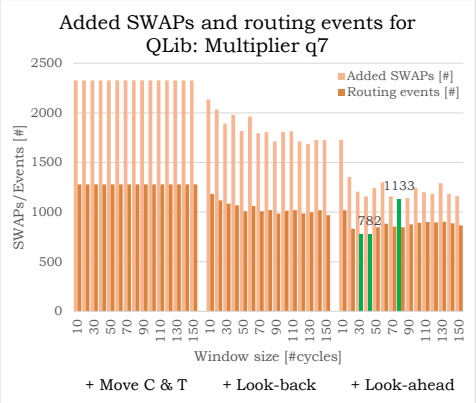


Figure E.16: Number of routing events and added SWAPs for different functions and window size for the QLib: Multiplier 7-qubit algorithm.

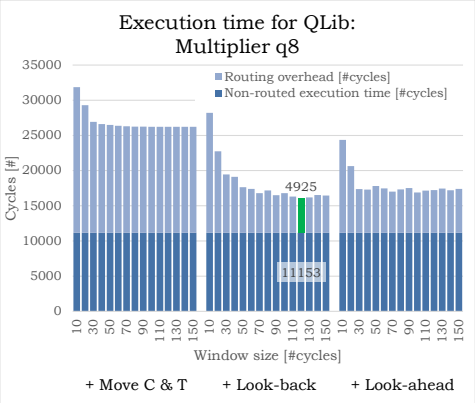


Figure E.17: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: Multiplier 8-qubit algorithm.

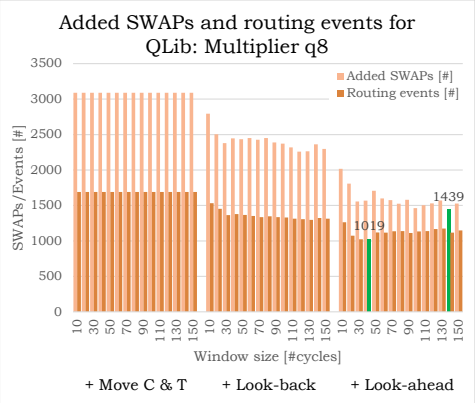


Figure E.18: Number of routing events and added SWAPs for different functions and window size for the QLib: Multiplier 8-qubit algorithm.

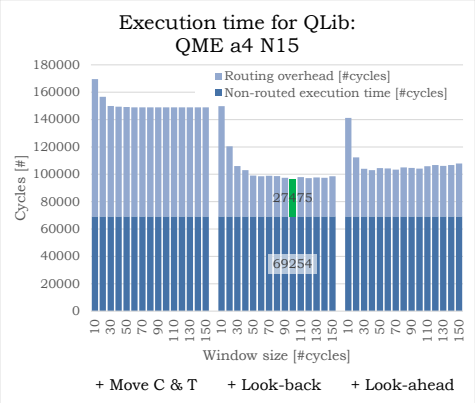


Figure E.19: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: QME a4 N15 algorithm.

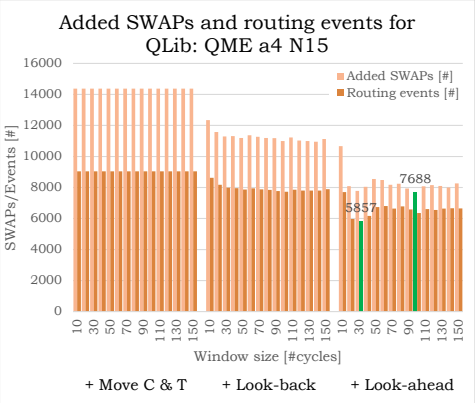


Figure E.20: Number of routing events and added SWAPs for different functions and window size for the QLib: QME a4 N15 algorithm.

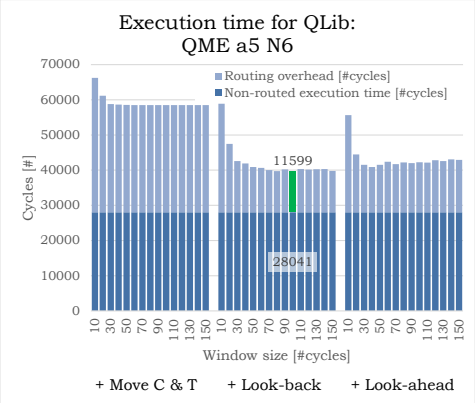


Figure E.21: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: QME a5 N6 algorithm.

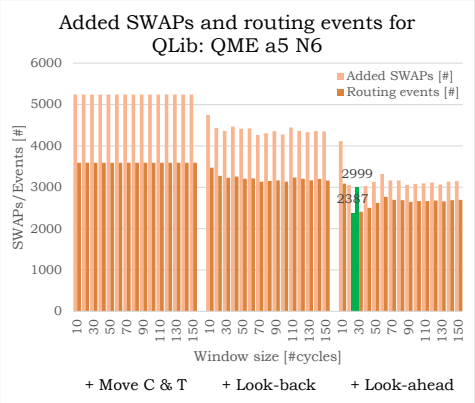


Figure E.22: Number of routing events and added SWAPs for different functions and window size for the QLib: QME a5 N6 algorithm.

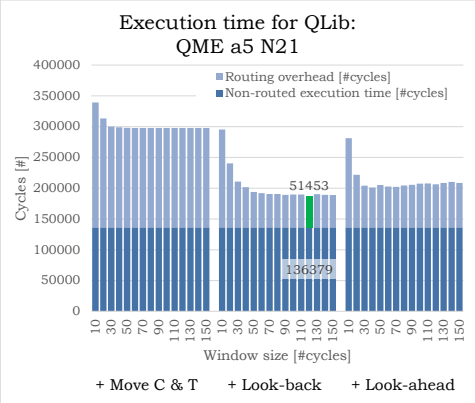


Figure E.23: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: QME a5 N21 algorithm.

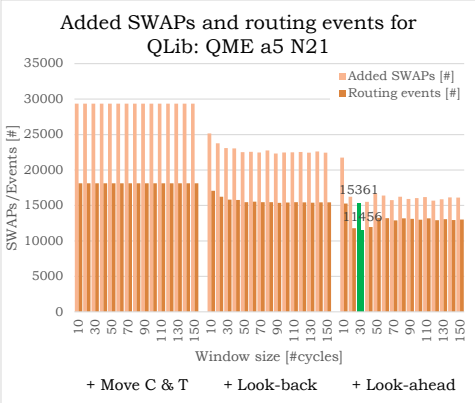


Figure E.24: Number of routing events and added SWAPs for different functions and window size for the QLib: QME a5 N21 algorithm.

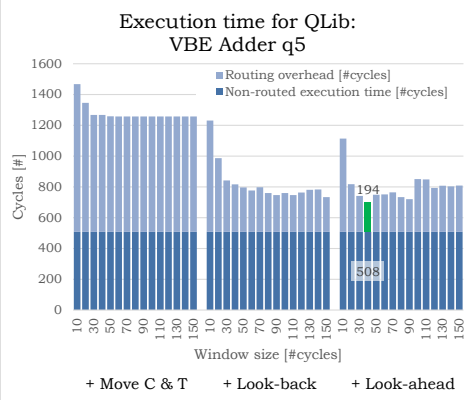


Figure E.25: Quantum circuit execution time in cycles for different functions and window sizes for the QLib: VBE Adder 5-qubit algorithm.

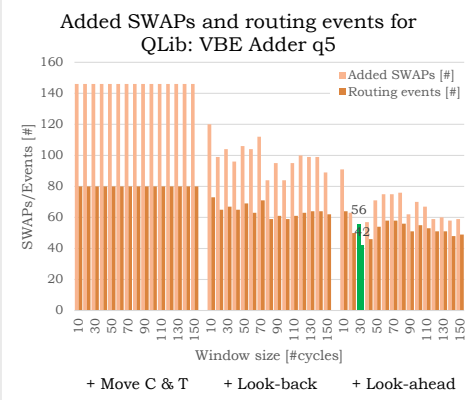


Figure E.26: Number of routing events and added SWAPs for different functions and window size for the QLib: VBE Adder 5-qubit algorithm.

F

Routing algorithm execution time for random benchmarks

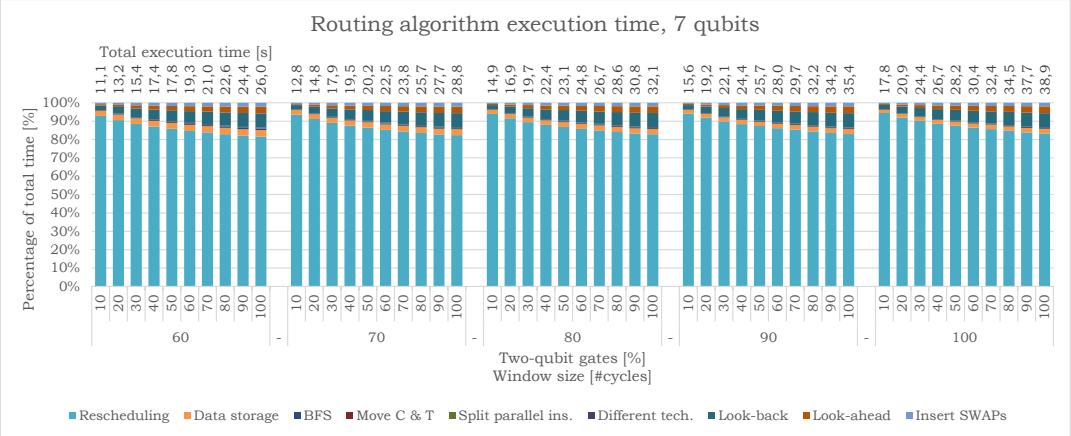


Figure F.1: The execution time of the routing algorithm for a 10000, 7 qubits, a random generated circuit for different percentage of two-qubit gates, 10-100% (the different bars) for a window size of 60-100[#cycles]. The contribution (in percentage) to the total time of the different functions is shown.

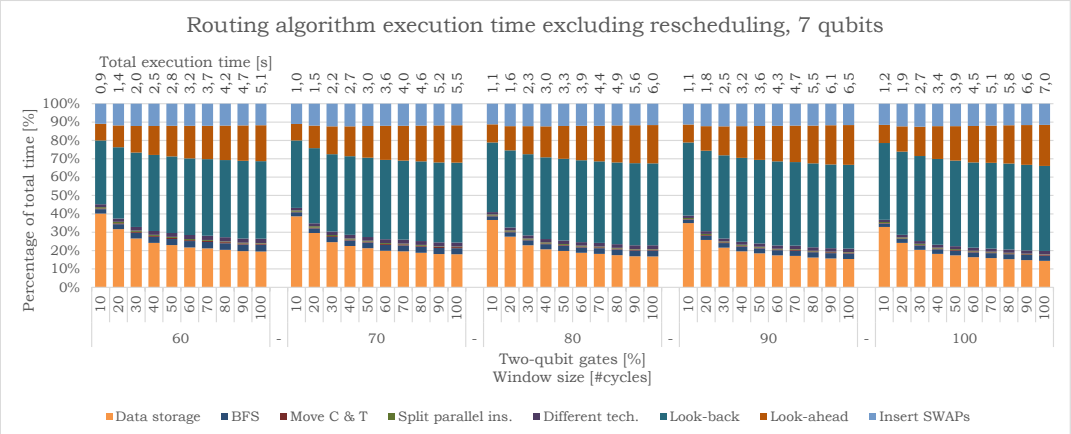


Figure F.2: The execution time of the routing algorithm for a 10000, 7 qubits, a random generated circuit for different percentage of two-qubit gates, 10-100% (the different bars) for a window size of 60-100[#cycles]. The contribution (in percentage) to the total time excluding rescheduling of the different functions is shown.

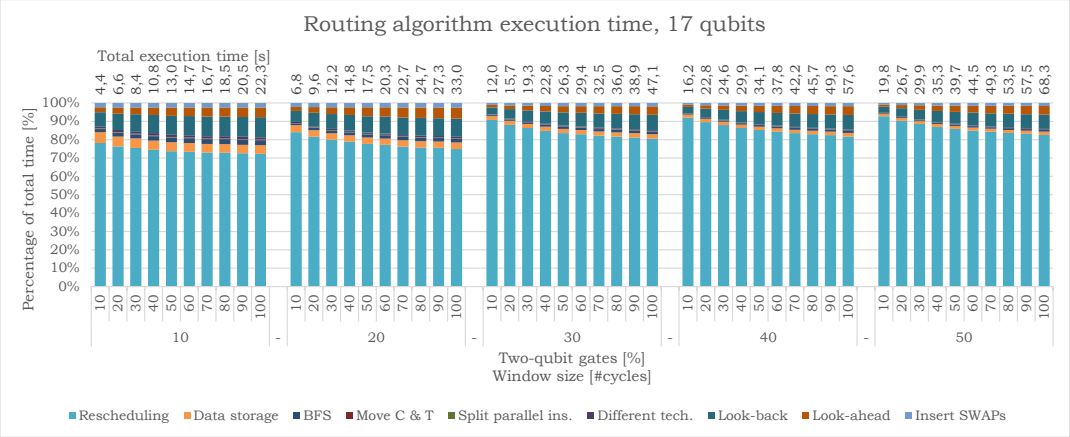


Figure F.3: The execution time of the routing algorithm for a 10000, 17 qubits, a random generated circuit for different percentage of two-qubit gates, 10-100% (the different bars) for a window size of 10-50[#cycles]. The contribution (in percentage) to the total time of the different functions is shown.

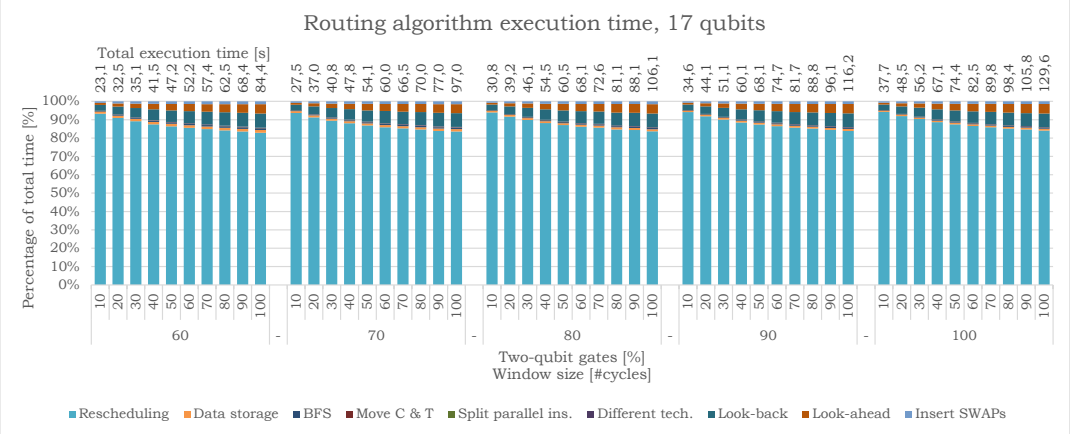


Figure F.4: The execution time of the routing algorithm for a 10000, 17 qubits, a random generated circuit for different percentage of two-qubit gates, 10-100% (the different bars) for a window size of 60-100[#cycles]. The contribution (in percentage) to the total time of the different functions is shown.

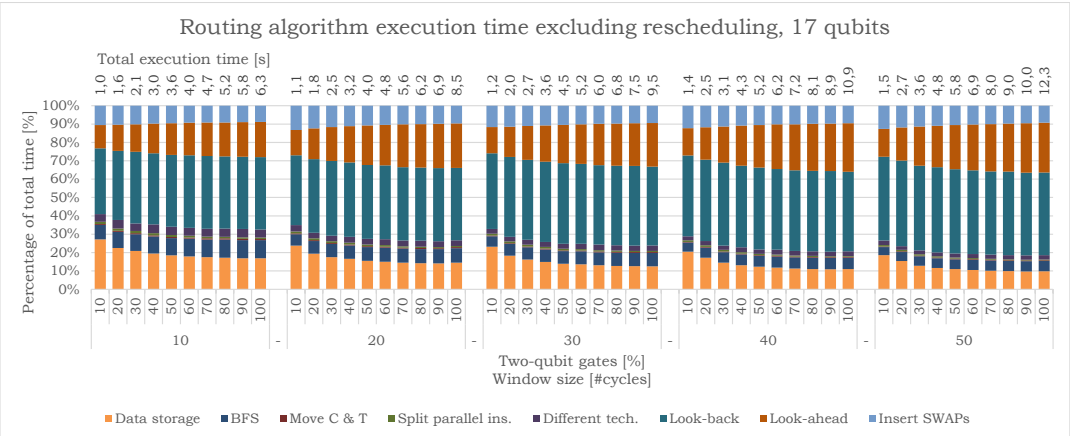


Figure F.5: The execution time of the routing algorithm for a 10000, 17 qubit, a random generated circuit for different percentage of two-qubit gates, 10-50%. The contribution (in percentage) to the total time excluding rescheduling of the different functions is shown.

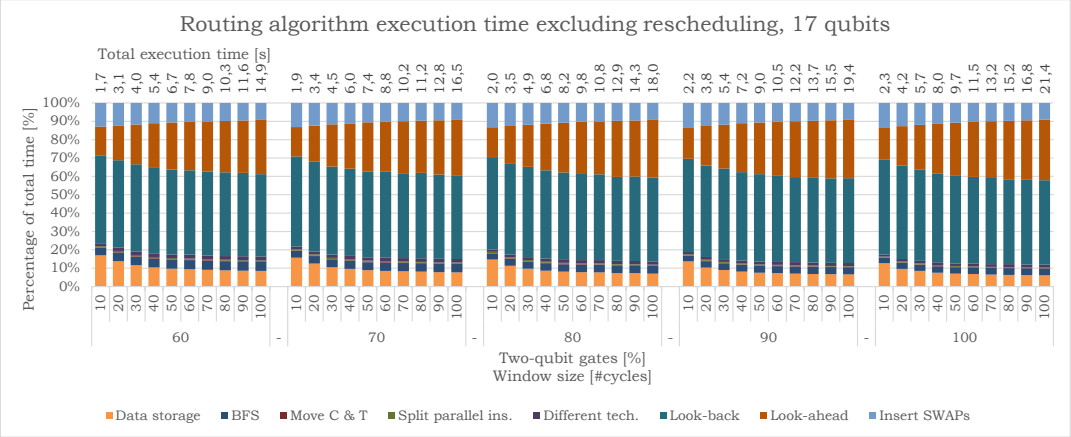


Figure F.6: The execution time of the routing algorithm for a 10000, 17 qubits, a random generated circuit for different percentage of two-qubit gates, 10-100% (the different bars) for a window size of 60-100[#cycles]. The contribution (in percentage) to the total time excluding rescheduling of the different functions is shown.

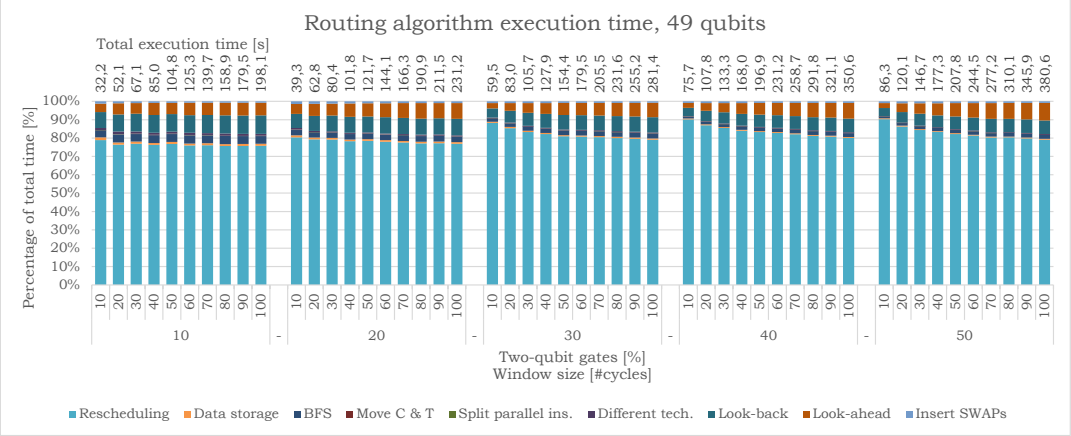


Figure F.7: The execution time of the routing algorithm for a 10000, 49 qubits, a random generated circuit for different percentage of two-qubit gates, 10-100% (the different bars) for a window size of 60-100[#cycles]. The contribution (in percentage) to the total time of the different functions is shown.

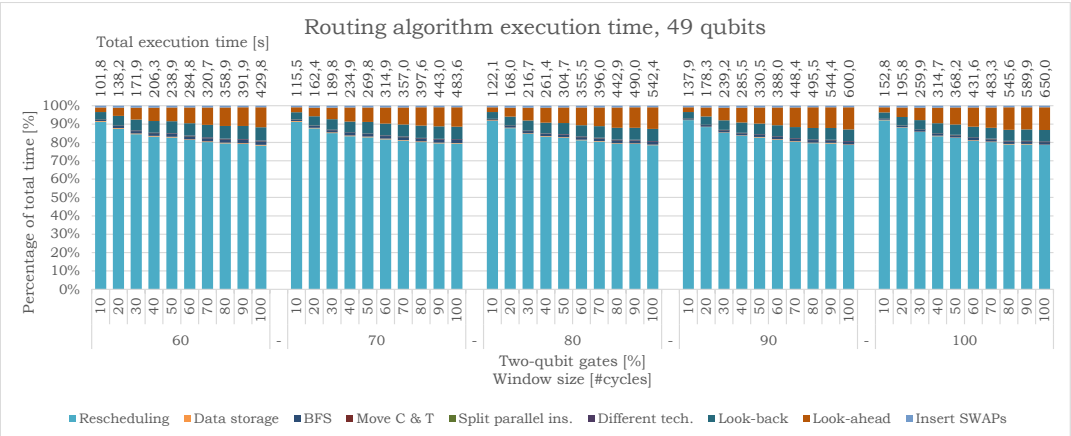


Figure F.8: The execution time of the routing algorithm for a 10000, 49 qubits, a random generated circuit for different percentage of two-qubit gates, 10-100% (the different bars) for a window size of 60-100[#cycles]. The contribution (in percentage) to the total time of the different functions is shown.

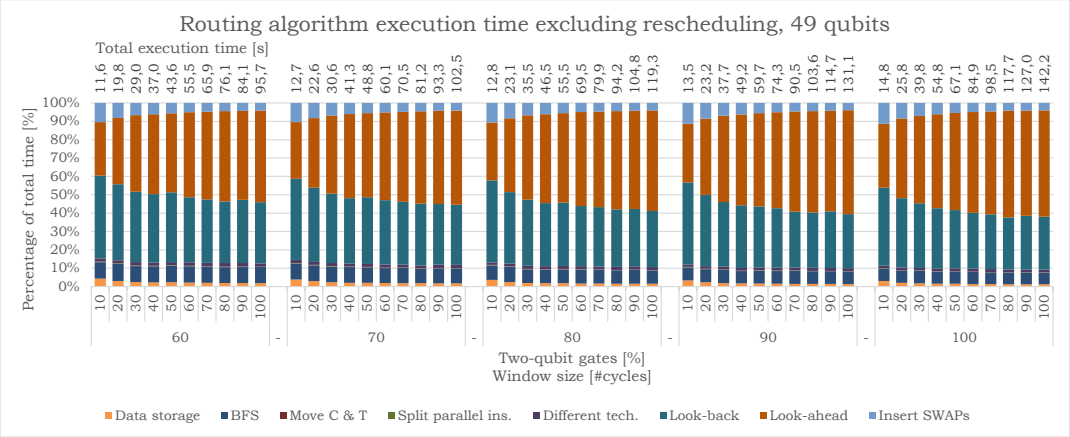


Figure F.9: The execution time of the routing algorithm for a 10000, 49 qubits, a random generated circuit for different percentage of two-qubit gates, 10-100% (the different bars) for a window size of 60-100[#cycles]. The contribution (in percentage) to the total time excluding rescheduling of the different functions is shown.

Bibliography

[1] Mohammad Gh Alfailakawi, Imtiaz Ahmad, and Suha Hamdan. Harmony-search algorithm for 2d nearest neighbor quantum circuits realization. *Expert Systems with Applications*, 61:16–27, 2016.

[2] Carmen G Almudever, L Lao, Xiang Fu, N Khammassi, Imran Ashraf, Dan Iorga, S Varsamopoulos, C Eichler, A Wallraff, L Geck, et al. The engineering challenges in quantum computing. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 836–845. IEEE, 2017.

[3] Debjyoti Bhattacharjee and Anupam Chattopadhyay. Depth-optimal quantum circuit placement for arbitrary topologies. *arXiv preprint arXiv:1703.08540*, 2017.

[4] Dean Copsey, Mark Oskin, Francois Impens, Tzvetan Metodiev, Andrew Cross, Frederic T Chong, Isaac L Chuang, and John Kubiatowicz. Toward a scalable, silicon-based quantum computing architecture. *IEEE Journal of selected topics in quantum electronics*, 9(6):1552–1569, 2003.

[5] Mohammad Javad Dousti and Massoud Pedram. Minimizing the latency of quantum circuits during mapping to the ion-trap circuit fabric. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 840–843. EDA Consortium, 2012.

[6] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

[7] X Fu, MA Rol, CC Bultink, J van Someren, N Khammassi, I Ashraf, RFL Vermeulen, JC de Sterke, WJ Vlothuizen, RN Schouten, et al. An experimental microarchitecture for a superconducting quantum processor. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 813–825. ACM, 2017.

[8] Xiang Fu, L Riesebos, Lingling Lao, Carmen G Almudever, Fabio Sebastiano, R Versluis, Edoardo Charbon, and Koen Bertels. A heterogeneous quantum computer architecture. In *Proceedings of the ACM International Conference on Computing Frontiers*, pages 323–330. ACM, 2016.

[9] Alexander S Green, Peter LeFanu Lumsdaine, Neil J Ross, Peter Selinger, and Benoît Valiron. Quipper: a scalable quantum programming language. In *ACM SIGPLAN Notices*, pages 333–342. ACM, 2013.

[10] Shima Hassanpour and Monireh Houshmand. Bidirectional quantum teleportation via entanglement swapping. In *Electrical Engineering (ICEE), 2015 23rd Iranian Conference on*, pages 501–503. IEEE, 2015.

[11] Clare Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

[12] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T Chong, and Margaret Martonosi. Scaffcc: Scalable compilation and analysis of quantum programs. *Parallel Computing*, 45: 2–17, 2015.

[13] N Khammassi. Openql, high-level quantum programming language. Not published, 2017.

[14] N Khammassi and K Bertels. Towards a standardized quantum assembly language qasm 1.0. Unpublished, date of consulting: 2017-10-01, 2017.

[15] Nader Khammassi, I Ashraf, X Fu, Carmen G Almudever, and Koen Bertels. Qx: A high-performance quantum computer simulation platform. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 464–469. IEEE, 2017.

[16] Richard E Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

[17] Anurag Kumar, D Manjunath, and Joy Kuri. *Communication networking: an analytical approach*. Elsevier, 2004.

[18] Samuel A Kutin, David Petrie Moulton, and Lawren M Smithline. Computation at a distance. *arXiv preprint quant-ph/0701194*, 2007.

[19] Lao L et al. Simulation of fault-tolerant operations on rotated planar surface codes. Unpublished, date of consulting: 2017-11-01, 2017.

[20] Chia-Chun Lin, Amlan Chakrabarti, and Niraj K Jha. Qlib: Quantum module library. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 11(1):7, 2014.

[21] Chia-Chun Lin, Susmita Sur-Kolay, and Niraj K Jha. Paqcs: Physical design-aware fault-tolerant quantum circuit synthesis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(7):1221–1234, 2015.

[22] Aaron Lye, Robert Wille, and Rolf Drechsler. Determining the minimal number of swap gates for multi-dimensional nearest neighbor quantum circuits. In *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, pages 178–183. IEEE, 2015.

[23] Tzvetan S Metodi, Darshan D Thaker, Andrew W Cross, Frederic T Chong, and Isaac L Chuang. Scheduling physical operations in a quantum information processor. In *Proceedings of SPIE*, volume 6244, pages 62440T–62440T, 2006.

[24] Michael A Nielsen and Isaac L Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2004.

[25] Michal Pióro and Deep Medhi. *Routing, flow, and capacity design in communication and computer networks*. Elsevier, 2004.

[26] Ville Rantala, Teijo Lehtonen, Juha Plosila, et al. *Network on chip routing algorithms*. Turku Centre for Computer Science, 2006.

[27] Mehdi Saeedi, Robert Wille, and Rolf Drechsler. Synthesis of quantum circuits for linear nearest neighbor architectures. *Quantum Information Processing*, 10(3):355–377, 2011.

[28] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures. In *Proceedings of the 50th Annual Design Automation Conference*, page 41. ACM, 2013.

[29] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. Qubit placement to minimize communication overhead in 2d quantum architectures. In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, pages 495–500. IEEE, 2014.

[30] Yu Tomita and Krysta M Svore. Low-distance surface codes under realistic quantum noise. *Physical Review A*, 90(6):062320, 2014.

[31] R Versluis, S Poletto, N Khammassi, N Haider, DJ Michalak, A Bruno, Koen Bertels, and Leo DiCarlo. Scalable quantum circuit and control for a superconducting surface code. *arXiv preprint arXiv:1612.08208*, 2016.

[32] Mark Whitney, Nemanja Isailovic, Yatish Patel, and John Kubiatowicz. Automated generation of layout and control for quantum circuits. In *Proceedings of the 4th international conference on Computing frontiers*, pages 83–94. ACM, 2007.

[33] Robert Wille, Aaron Lye, and Rolf Drechsler. Optimal swap gate insertion for nearest neighbor quantum circuits. In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, pages 489–494. IEEE, 2014.

[34] Robert Wille, Oliver Keszocze, Marcel Walter, Patrick Rohrs, Anupam Chattopadhyay, and Rolf Drechsler. Look-ahead schemes for nearest neighbor optimization of 1d and 2d quantum circuits. In *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*, pages 292–297. IEEE, 2016.

[35] Alwin Zulehner, Stefan Gasser, and Robert Wille. Exact global reordering for nearest neighbor quantum circuits using A*. In *International Conference on Reversible Computation*, pages 185–201. Springer, 2017.