Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Bringing the Concepts of Virtualization to Gate-based Quantum Computing

Marvin Bechtold

Course of Study:	Informatik
Examiner:	Prof. Dr. Dr. h. c. Frank Leymann
Supervisor:	Daniel Vietz, M.Sc.
Commenced:	November 20, 2020
Completed:	May 20, 2021

Abstract

Quantum computing is a promising paradigm to solve certain computational problems that are intractable for classical computers. Quantum computers manipulate bits of quantum information called qubits. Their development has gained momentum in recent years. Cloud providers offered the first quantum computers for commercial use via the cloud and continue to expand their quantum computing service offerings. Classical computing in the cloud uses virtualization to abstract from the physical computing resources to virtual resources. It enables resource sharing among multiple clients and dynamic allocation of resources to clients and applications. To this end, physical resources can be aggregated and partitioned to create customized virtual resources. In contrast, the logical execution of a quantum circuit that illustrates the computation is still tightly bound to the physical quantum device. Cloud users need to select a suitable physical quantum computer for their computations. If the calculation requires more qubits than the quantum computer has, the device cannot execute it. Otherwise, the execution of the quantum circuit blocks the entire quantum machine regardless of the qubit capacity used. This work develops a virtualization concept for quantum computing to separate the logical quantum computation from the underlying hardware. Aggregation and partitioning of quantum circuits decouple the one-to-one relationship between the computation of a quantum circuit and the execution hardware. The goal is to automate the mapping between the quantum circuits and appropriate physical quantum execution resources, including their aggregation and partitioning. The concept is implemented in a prototype and evaluated with modern quantum computers.

Contents

List of Figures

List of Tables

List of Listings

Acronyms

BV Bernstein-Vazirani. 62

EPC Error per Clifford. 54

HWEA Hardware efficient ansatz. 62

IBM QX IBM Quantum Experience. 41

NISQ Noisy Intermediate-Scale Quantum. 15, 24

QCaaS Quantum Computing as a Service. 25

QCSP Quantum Computing Service Platform. 41

QER Quantum execution resource. 15

QFT Quantum Fourier Transformation. 62

QPU Quantum processing unit. 15

RB Randomized benchmarking. 53

UCCSD Unitary Coupled Cluster with Singles and Doubles. 63

1 Introduction

The conception of the quantum computer is a computing device that operates according to the laws of quantum mechanics. Over 40 years ago, Benioff [Ben80] described the first quantum mechanical model of a computer. The theoretical promise of quantum computers is to efficiently solve problems that are intractable for classical computers [NC10]. In 1982, Feynman [Fey82] conceived the idea of using a quantum computer to efficiently simulate the evolution of a quantum system that seems impossible on a classical computer. Nevertheless, there was no physical realization of a quantum computer, and it was questionable whether there ever would be one. Despite the lack of a physical machine, researchers developed the first algorithms that exploit the conceptual power of quantum computing, such as Grover's algorithm for unstructured search [Gro96] or Shor's algorithm for factorizing numbers [Sho99]. Around the turn of the millennium, scientists realized the first implementation of quantum algorithms on physical quantum hardware [CGK98; KMSW00]. Today's quantum computers are called Noisy Intermediate-Scale Quantum (NISQ) machines [Pre18]. Their Quantum processing units (QPUs) provide a limited number of noisy and error-prone qubits.

In recent years, the first NISQ computers have been offered for commercial use via the cloud [Cas17]. It was a crucial step in providing access to quantum computers to a wide audience of researchers and companies and facilitating further development of the technology [MRN+17]. In addition to physical NISQ devices, cloud providers also offer simulators of QPUs. The term Quantum execution resource (QER) combines these two ways of executing a quantum program. To perform a computation on a cloud-based QER, users must first select a resource. The QERs differ in their characteristics and capabilities. Second, the user sends an executable, a quantum circuit, to the selected QER. The QER computes the quantum circuit. After execution, the cloud service returns the result to the user.

Computation time on a QER, especially on a physical QPU, is a scarce resource. To manage multiple users' access to the QPUs, the cloud providers use a time-based resource-sharing model. The model temporarily assigns QPUs to users for their computation. During this time window, the QPU exclusively processes the user's quantum circuit. Other users with pending execution tasks must wait until a QPU is fully available. Thus, one QPU of the cloud provider executes exactly one quantum circuit at a time. It follows that the number of qubits of the QPU constrains the size of the quantum circuit and, conversely, the execution of one quantum circuit blocks an entire QPU regardless of the qubit utilization. The one-to-one coupling between the execution of a quantum circuit and the underlying QER is a problem because it prevents an efficient allocation of resources.

Virtualization plays an essential role in classical computing as an enabler for cloud computing [Zha18]. It separates logical functionality from physical realization [Scr17], providing the ability to share computing resources and dynamically map virtual resources to clients and applications as needed [Zha18]. Virtualization is thus able to free hardware resources from the constraints of resource allocation in the time-based resource-sharing model. It facilitates resource sharing by aggregation and partitioning. For instance, virtualization allows multiple users to share a large physical server by partitioning it into independent virtual resources and combining multiple disks into one virtual cloud storage.

The objective of this thesis is to develop a concept of virtualization for quantum computing in order to separate the logical computation of a quantum circuit from the underlying physical hardware realization. To this end, an intermediate software layer is introduced to provide a unified interface for computing logical quantum circuits over existing quantum hardware. The intermediary routine automatically maps the quantum circuit to the underlying QERs and executes it. Techniques that allow aggregation and partitioning of the actual quantum resources are of particular interest. However, the user should overlook the virtualization process and get the execution results as usual.

The remainder of the thesis is organized as follows. Chapter 2 covers the fundamentals of quantum computation and virtualization. Subsequently, Chapter 3 summarizes the current situation, presents the problem in detail, and formulates the research objective of this thesis. Chapter 4 gives an overview of related work and places it in the context of this thesis. In the following, Chapter 5 presents the developed virtualization concept for quantum computing. Chapter 6 presents a prototype implementation of the concept. Chapter 7 assesses the feasibility of the concept on the basis of the implementation. Subsequently, Chapter 8 discusses the result and shows the advantages and limitations of the presented solution. Finally, Chapter 9 concludes this thesis and provides an outlook.

2 Fundamentals

This chapter forms the basis for the rest of the thesis. It establishes the mathematical concept of quantum computing. We then address the physical implementation of quantum computers and current quantum cloud offerings. Finally, the concept of virtualization from classical computing is concisely highlighted.

2.1 Quantum Computation

Quantum computation uses the principles of quantum mechanics to perform computations. A device that performs quantum computations is called a quantum computer. Richard Feynman was one of the first to come up with the concept of a quantum computer in 1982 to efficiently simulate other quantum systems [Fey82]. David Deutsch then extended this idea in 1985 by defining the quantum Turing machine and stating that a quantum computer can provide a computational advantage over the classical computer [Deu85]. Furthermore, Deutsch introduced the quantum circuit model in 1989 to represent quantum computations [Deu89]. The following section briefly highlights the necessary mathematical concepts and abstractions of gate-based quantum computation and the state of physical implementation. Besides the gate-based approach, there are also the one-way [RB01] and adiabatic [ADK+08] quantum computation approaches. In this work, only the gate-based model is considered. Unless otherwise referenced, the core of this section is based on the standard work by Nielsen and Chuang [NC10].

2.1.1 Qubit

In classical computation, a bit is the basic unit of information. It can be in one of two possible states: either 0 or 1. Analogous to the bit in classical computation, the quantum bit, or qubit, is the basic unit of quantum information. The corresponding states of a qubit in the Dirac notation are $|0\rangle$ and $|1\rangle$, and their vector representation in the standard basis is

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{2.1}$$

Unlike the classical bit, which has only these two possible states, a qubit has infinitely many. The state-space of the qubit is a two-dimensional complex vector space with an inner product called a two-dimensional Hilbert space. The states $|0\rangle$ and $|1\rangle$ are the computational basis states and form an orthonormal basis $\{|0\rangle, |1\rangle\}$ for the state-space $\mathbb{H}\,(\cong \mathbb{C}^2)$. The state vector of a qubit is a unit vector in the state-space $\mathbb{H}$ and describes the state completely:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{2.2}$$

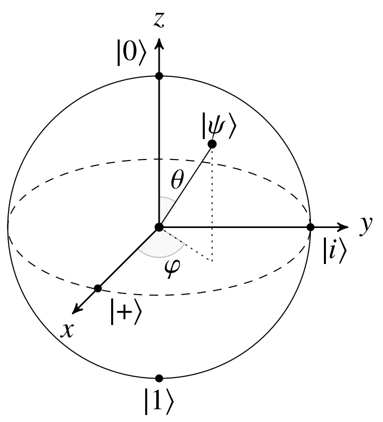


Figure 2.1: The state of a qubit on the Bloch sphere

The coefficients α and β are called amplitudes, and $\alpha, \beta \in \mathbb{C}$. Due to $|\,|\psi\rangle\,| = 1$, α and β must be bounded by the equation

$$|\alpha|^2 + |\beta|^2 = 1. \tag{2.3}$$

A characteristic phenomenon of a qubit is superposition. It occurs when α and β are nonzero and the state of the qubit is an overlap of $|0\rangle$ and $|1\rangle$.

Given Equation (2.3), the following expression gives an alternative representation of the state of the qubit:

$$|\psi\rangle = e^{i\gamma}\left(\cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle\right) \tag{2.4}$$

where $\theta \in [0, \pi]$ and $\varphi \in [0, 2\pi]$. Since global phase factors are not measurable and therefore have no observable effects, ignoring the factor $e^{i\gamma}$ simplifies Equation (2.4) into

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle \tag{2.5}$$

This simplification allows a geometric interpretation of the qubit's state. The values θ and φ determine a point on a three-dimensional sphere, often called the Bloch sphere. Figure 2.1 shows the Bloch sphere and how the angles θ and φ define the $|\psi\rangle$ state.

In this representation, the computational basis states lie on the z-axis. The state $|0\rangle$ lies at the north pole of the Bloch sphere, the state $|1\rangle$ at the south pole. Other important quantum states with specific labels are the $|+\rangle$, $|-\rangle$, $|i\rangle$, and $|-i\rangle$ states. Their representation in the computational basis is as follows:

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{2.6}$$

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{2.7}$$

$$|i\rangle = \frac{1}{\sqrt{2}}(|0\rangle + i\,|1\rangle) \tag{2.8}$$

$$|-i\rangle = \frac{1}{\sqrt{2}}(|0\rangle - i\,|1\rangle) \tag{2.9}$$

The $|+\rangle$ and $|-\rangle$ states lie on the x-axis, and the $|i\rangle$ and $|-i\rangle$ states lie on the y-axis of the Bloch sphere. Each of these state pairs forms an orthonormal basis of the qubit's state-space.

Multiple qubits

As in classical computing, it is possible to combine several single qubits into one register. The tensor product of the state-space of the single qubits describes the state-space of the quantum register:

$$\mathbb{H}_{\text{reg}} = \mathbb{H}^{\otimes n} = \mathbb{H}_{n-1} \otimes ... \otimes \mathbb{H}_1 \otimes \mathbb{H}_0 \tag{2.10}$$

Each of the one-qubit state-spaces $\mathbb{H}_i$ contains two base states. Combining them into a n qubit register creates a state-space spanned by 2^n base states and results in a 2^n dimensional vector space.

Suppose we have two qubits. The two-qubit register has four computational base states:

$$|00\rangle = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \quad |01\rangle = \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, \quad |10\rangle = \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, \quad |11\rangle = \begin{pmatrix}0\\0\\0\\1\end{pmatrix} \tag{2.11}$$

Like the single qubit, the quantum register can also be in superposition. Therefore, the general state of a two-qubit register is as follows.

$$|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle = \begin{pmatrix}\alpha_{00}\\\alpha_{01}\\\alpha_{10}\\\alpha_{11}\end{pmatrix} \tag{2.12}$$

Entanglement is a quantum phenomenon involving multiple qubit registers. When the state of a quantum register can be decomposed into the tensor product of the individual qubits, it is called separable. However, this is not possible for the vast majority of quantum states [RP11]. These qubit registers cannot be considered as a pure combination of the individual qubits' states and are entangled. More formally, a quantum register of size n is entangled if there are no single qubit states $|x_0\rangle, ..., |x_{n-1}\rangle$ such that the register's state is representable as $|\psi\rangle = |x_0\rangle \otimes ... \otimes |x_{n-1}\rangle$. Entanglement is a unique property of quantum computing. It is necessary to exploit this property if the quantum algorithm is to provide an exponential speedup over classical computation [JL03].

The Bell states $\{\Phi^+, \Phi^-, \Psi^+, \Psi^-\}$ are an example of entangled two-qubit states and form an orthonormal basis for two-qubit state-space called the Bell basis. None of them can be represented as the tensor product of two single qubits. Their representation in the computational basis is as follows:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{2.13}$$

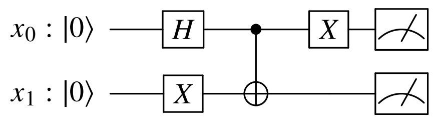
$$|\Phi^-\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) \tag{2.14}$$

$$|\Psi^+\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \tag{2.15}$$

$$|\Psi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) \tag{2.16}$$

2.1.2 Quantum circuits

In the gate-based quantum computing model, quantum circuits are a graphical framework to describe a quantum computation. Quantum circuits are composed of gates, wires, and measurements. A gate represents an operation on the qubits of a quantum computer. It takes as input one or more qubits and outputs the manipulated qubits. The output number of qubits must always be equal to the input number. A wire represents a qubit. Following a wire past gates from left to right describes the transformations applied to the associated qubit. Generally, a measurement is placed at the end of the wire. In the following, an exemplary quantum circuit with two qubits is shown.



(2.17)

Both qubits are initialized to the $|0\rangle$ state. Then a series of one- and two-qubit gates modify the qubits. At the end, the state of the two qubits is measured. Two characteristics of a quantum circuit are its width and its depth or length. The width is the number of qubits, and the depth is the length of the longest sequence of operations on a qubit. In the above quantum circuit, the width is two, and the depth is three. The operations of gates and measurements are described in more detail below.

2.1.3 Operations

Performing computations on a quantum computer means applying unitary transformations to its qubits. A unitary matrix U is a square matrix and describes the operation. The matrix-vector product computes the state that results from performing the operation U on the state $|\psi\rangle$:

$$|\psi\rangle \longrightarrow U\,|\psi\rangle \tag{2.18}$$

A matrix U is unitary if $U^\dagger U = I$, where $U^\dagger$ is the adjoint of U, and I is the identity matrix. The adjoint of a matrix U is the transpose of the element-wise complex conjugate matrix of U, or $U^\dagger = (\overline{U})^T$ for short. Unitary transformations preserve the inner product of two vectors. Therefore, they are length-preserving and keep the resulting quantum state vectors normalized: $|U\,|\psi\rangle\,| = 1$. Therefore, a unitary matrix U preserves the validity of the quantum state after its application. Another essential property of the unitary matrix is that $U^\dagger = U^{-1}$. Thus, any unitary matrix is invertible, and thus any operation on a gate-based quantum computer is reversible. In the following sections, the single and multiple qubit operations are explained in more detail.

Single qubit gates

Unlike classical computing, where the NOT operation is the only single bit operation, in quantum computing, there are infinitely many single-qubit operations. Each 2×2 unitary matrix describes a gate manipulating a single qubit. Table 2.1 gives an overview of frequently used single-qubit transformations. To apply a single-qubit operation U_i to the i-th qubit of a quantum register of size n, the following transformation must be applied to the entire quantum register [RP11]:

$$U = I_1 \otimes \ldots \otimes I_{i-1} \otimes U_i \otimes I_{i+1} \otimes \ldots \otimes I_n \tag{2.19}$$

Name	Symbol	Gate	Dirac notation	Matrix
Identity	I	$-\boxed{I}-$	$\|0\rangle\langle 0\| + \|1\rangle\langle 1\|$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
Pauli-X	X	$-\boxed{X}-$	$\|1\rangle\langle 0\| + \|0\rangle\langle 1\|$	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Pauli-Y	Y	$-\boxed{Y}-$	$i\,\|1\rangle\langle 0\| - i\,\|0\rangle\langle 1\|$	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Pauli-Z	Z	$-\boxed{Z}-$	$\|0\rangle\langle 0\| - \|1\rangle\langle 1\|$	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
Hadamard	H	$-\boxed{H}-$	$\frac{1}{\sqrt{2}}(\|0\rangle\langle 0\| + \|1\rangle\langle 0\| + \|0\rangle\langle 1\| + \|1\rangle\langle 1\|)$	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
Phase	S	$-\boxed{S}-$	$\|0\rangle\langle 0\| + i\,\|1\rangle\langle 1\|$	$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$

Table 2.1: Single qubit gates. Matrix representations are with respect to the standard basis.

The most commonly used single-qubit transformations are the Pauli transformations I, X, Y, and Z. Applying the Pauli matrices to a single qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ has the following effect:

$$I(|\psi\rangle) = \alpha|0\rangle + \beta|1\rangle = |\psi\rangle \tag{2.20}$$

$$X(|\psi\rangle) = \beta|0\rangle + \alpha|1\rangle \tag{2.21}$$

$$Y(|\psi\rangle) = i\beta|0\rangle - i\alpha|1\rangle \tag{2.22}$$

$$Z(|\psi\rangle) = \alpha|0\rangle - \beta|1\rangle \tag{2.23}$$

Pauli-X is negation, Pauli-Z is a relative phase change, and Pauli-Y is a combination of negation and phase change with $Y = iXZ$ [RP11]. The identity operation is often omitted in the list of Pauli matrices.

Another important single-qubit gate is the Hadamard transformation. The Hadamard gate transforms the computational basis to the Hadamard basis.

$$H|0\rangle = |+\rangle \tag{2.24}$$

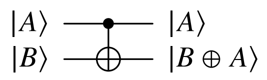
$$H|1\rangle = |-\rangle \tag{2.25}$$

Applying the Hadamard transformation to one of the computational basis states yields an even superposition of $|0\rangle$ and $|1\rangle$ with $|\alpha|^2 = |\beta|^2$.

Multiple qubit gates

Multiple-qubit gates are operations that require at least two qubits as input. Multi-qubit transformations generated by the tensor product of single-qubit operations are not interesting since they are equivalent to performing the single-qubit transformation on each of the qubits individually in some order [RP11].

A typical gate that single-qubit gates cannot replace is the controlled-NOT or CNOT gate. It works with two qubits. One qubit acts as the control qubit, the other as the target qubit. The input of the control qubit determines whether the input of the target qubit is negated. The control qubit remains unchanged. The CNOT operation is shown on the left as a gate in a quantum circuit, and the corresponding unitary matrix is shown on the right. Since the CNOT is a two-qubit operation, the matrix is of size 4×4.



(2.26)

$$U_{CN} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.27}$$

2.1.4 Measurement

The amplitudes α and β define a qubit's state in the computational basis. However, a measurement of the qubit does not provide the amplitudes. Instead, the measurement is a random experiment that yields either a 0 with probability $|\alpha|^2$ or a 1 with probability $|\beta|^2$. Moreover, the measurement process changes the state, and the qubit's superposition of $|0\rangle$ and $|1\rangle$ collapses to the specific state, consistent with the measurement result.

More generally, a measurement operator describes the measurement of a quantum system. It is a Hermitian operator acting on the quantum system to be measured. A matrix A is Hermitian if and only if $A = A^\dagger$. The matrix $A^\dagger$ is the adjoint of A, which is the transpose of the element-wise complex conjugated matrix of A, or $A^\dagger = (\overline{A})^T$ for short. All eigenvalues of a Hermitian operator are real. In the special case of projective measurement, which is sufficient for our purposes, the measurement is described by an observable O, which is a Hermitian operator that has a spectral decomposition,

$$O = \sum_m m P_m \tag{2.28}$$

where P_m is the projector onto the eigenspace of O with eigenvalue m. Assuming any quantum state $|\psi\rangle$, then the probability of obtaining m by measuring with O is

$$p(m) = \langle\psi| P_m |\psi\rangle \tag{2.29}$$

where $\langle\psi|$ is the transpose of $|\psi\rangle$. Given that the measurements yields m, the new state $|\psi'\rangle$ of the quantum system is

$$|\psi'\rangle = \frac{P_m |\psi\rangle}{\sqrt{p(m)}}. \tag{2.30}$$

The expected value of the measurement of the state $|\psi\rangle$ with observable O is

$$E_\psi(O) = \sum_m m p(m) = \langle\psi| O |\psi\rangle = \langle O\rangle_\psi \tag{2.31}$$

The above example of the computational basis measurement can be achieved by choosing Pauli-Z as the observable. The eigenvectors of Pauli-Z are $|0\rangle$ and $|1\rangle$ with corresponding eigenvalues 1 and -1. Thus, measuring a qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ yields the measurement $m = 1$ with probability

$$p(1) = \langle\psi| |0\rangle \langle 0| |\psi\rangle = |\langle 0| |\psi\rangle|^2 = |\alpha|^2 \tag{2.32}$$

and the measurement $m = -1$ with probability

$$p(-1) = \langle\psi| |1\rangle \langle 1| |\psi\rangle = |\langle 1| |\psi\rangle|^2 = |\beta|^2. \tag{2.33}$$

2.1.5 Density operator

So far, we have used the state vector formulation for quantum computation. A mathematically equivalent formulation is the density operator or density matrix. The quantum state encoded as $|\psi\rangle$ in the state vector formulation is represented in the density operator language as the following matrix:

$$|\psi\rangle\langle\psi| \tag{2.34}$$

This is a pure quantum state. One advantage of the density operator is that it provides a much more convenient language for describing mixed quantum states. A mixed quantum state $\{p_i, |\psi_i\rangle\}$ is a statistical ensemble of pure quantum states. This means that the quantum system is in the pure state $|\psi_i\rangle$ with probability p_i, where $p_i > 0$ and $\sum_i p_i = 1$. Note that a mixed quantum state is different from a quantum superposition. Each of the pure quantum states can itself be in superposition. The density operator ρ, which describes the mixed state of the quantum system, is defined as follows.

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i| . \tag{2.35}$$

The density operator is convenient for mixed states because any state, pure or mixed, can be characterized by a single density matrix. Suppose we apply a unitary operator U to the mixed state. If the system was initially in state $|\psi_i\rangle$ with probability p_i, it is now in state $U|\psi_i\rangle$ with probability p_i. The equation describes the transformation by the unitary operator U

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i| \xrightarrow{U} \sum_i p_i U |\psi_i\rangle\langle\psi_i| U^\dagger = U\rho U^\dagger . \tag{2.36}$$

In the density operator formulation, the measurements are also described by an observable $O = \sum_m m P_m$. If the system is in state $|\psi_i\rangle$, then the probability of the measurement m is

$$p(m|i) = \langle\psi_i| P_m |\psi_i\rangle = \mathrm{Tr}(P_m |\psi_i\rangle\langle\psi_i|) \tag{2.37}$$

where Tr is the trace operator. Thus, measuring the state of the density operator ρ yields the measurement m with the following probability:

$$p(m) = \sum_i p(m|i) p_i = \mathrm{Tr}(P_m \rho) \tag{2.38}$$

This results in the following expected value for the measurement of state ρ with observable O:

$$E_\rho(O) = \sum_m m p(m) = \mathrm{Tr}\,(O\rho) \tag{2.39}$$

2.1.6 Quantum computers

So far, we have introduced the mathematical concept of quantum computation. But without a physical realization, quantum computation is just an idle curiosity that creates no value in applications. In order for a quantum device to be useful for performing calculations, DiVincenzo [DiV00] established the following five necessary requirements for a quantum computer:

1. A scalable physical system with well-characterized qubits

2. The possibility to initialize the state of the qubits to a simple fiducial state
3. Long relevant decoherence times, much longer than the gate operation time
4. A “universal” set of quantum gates
5. A qubit-specific measurement capability

However, current quantum computers have shortcomings and do not satisfactorily meet all five requirements. Today, quantum computers exist only with a limited number of up to several tens of qubits. Additionally, qubit connectivity, which allows operations between qubits, is limited in these systems. The number of qubits and their connectivity cannot be scaled up to build larger QPUs, which limits the first requirement.

Moreover, the quality of the qubits is limited. Noise that combines errors from different sources affects the qubits, and their state decays over time [LB20]. Errors occur when performing operations or measurements. Furthermore, qubits decohere due to unwanted interactions between them and their environment. These errors accumulate and interfere with the calculation. The noise of modern QPUs limits their operation time and conflicts with the third requirement. Preskill [Pre18] coined the term Noisy Intermediate-Scale Quantum (NISQ) devices for these quantum computers and expects that, despite the shortcomings, NISQ devices may in the near future be able to perform tasks that exceed the capabilities of current classical digital computers.

The term crosstalk addresses a subset of these NISQ errors. It describes the effect that a subsystem of a quantum machine - a qubit, measurement device, etc. - unintentionally has on another subsystem [SPR+20]. Crosstalk effects on QPUs cause crosstalk errors, which were reported by Sarovar et al. [SPR+20] as undesired dynamics that violate at least one of the following two principles:

- **Locality of operations** requires that the physical implementation of a quantum circuit does not create an unintended correlation between any qubits unless that circuit contains multi-qubit operations that intentionally couple them.
- **Independence of local operations** requires that an operation in a quantum circuit, such as a gate or a measurement, does not depend on what other operations (acting on disjoint qubits) occur in the circuit at the same time.

A QPU is free of crosstalk if its behavior in the execution of arbitrary quantum circuits satisfies the principle of locality and independence [SPR+20].

2.1.7 Cloud offerings

First cloud providers included NISQ devices in quantum computing services [Cas17]. Some of them have developed their hardware for this purpose. Well-known representatives are IBM or Rigetti. Others rely on third-party hardware, like Amazon or Microsoft. To run a quantum application on a cloud-hosted quantum computer, four steps are required [LBF+20]:

1. The respective platform must be installed on a classical computer
2. The quantum circuit must be realized with the installed platform
3. The quantum execution resource must be selected

4. The quantum circuit must be executed

Quantum Computing as a Service (QCaaS) describes the current service model where a QER, a quantum computer or simulator, is hosted in the cloud for quantum circuit execution (Item 4). Items 1 to 3 of the quantum application are integrated with classical applications and not in the cloud [LBF+20]. Cloud users need to take care of these issues themselves.

IBM has implemented a simple queue-based approach to accessing their public QERs in the cloud. Users create jobs in the queues, which are then processed one after the other by the corresponding QER. This approach is efficient for providing access to QERs. However, it has poor runtime efficiency due to the additional overhead of network connections and the use of a shared queue [KTP+20]. Rigetti seeks to improve runtime efficiency by accessing the exclusively reserved quantum computer via a virtual machine running on a classical computer cluster that resides on-premise. In both cases, time-based resource sharing allows multiple users to access scarce QPU resources. Each QPU is exclusively assigned to one customer for a finite period of time.

2.2 Virtualization

Virtualization does not refer to a single technology or methodology; instead, a large variety of tools and technologies are grouped under the term virtualization [Scr17]. There are server virtualization, memory virtualization, data virtualization, storage virtualization, network virtualization, and application virtualization, among others [Scr17]. Although these application areas of virtualization differ in their underlying tools and methods, they share similar concepts. The general concept common to all virtualization technologies is the separation of the logical from the physical [Scr17]. Virtualization introduces an abstracting software layer on top of existing physical computing resources, encapsulating their lower-level functionalities and enabling portability of higher-level functions based on them [SML10].

To achieve the abstracting separation, virtualization essentially consists of a mapping and emulating process [Zha18]: It maps the virtual resource to native hardware resource and then uses the latter for computation. This allows the virtual resources to be decoupled from the hardware. A virtual resource is not necessarily bound to a single existing hardware resource. Instead, the virtualization layer abstraction allows the aggregation or partitioning of the underlying physical computing resources into one or many resources in the virtual environment, including through the use of hardware and software partitioning or aggregation techniques [CN05].

The concept of resource aggregation is found in many virtualization technologies, such as in data or network virtualization. Data virtualization aggregates different underlying data resources and abstracts from their properties, including format or physical location [Scr17]. It provides other applications with a unified view of the entire dataset [Scr17]. Network virtualization can be used to aggregate multiple physical network infrastructures of heterogeneous architectures into a virtual network and provide end-to-end services in the aggregated virtual network across the different underlying physical networks [CB10].

Network virtualization not only enables the concept of resource aggregation but also the concept of resource partitioning. Virtual networks can divide an extensive physical network infrastructure into logically separated networks [CB10]. Furthermore, resource partitioning can also be found, in server virtualization, among other applications. The essential components of server virtualization are a hypervisor and virtual machines [Dan09]. The hypervisor forms an abstraction layer on top of the underlying physical server hardware. Virtual machines are efficient, isolated duplicates of real machines [PG74]. The hypervisor provides the management functionality to run multiple, logically separated virtual machines while sharing the server's hardware resources [Dan09]. Server virtualization thus enables the partitioning of a physical server into several logical instances, the virtual machines.

Virtualization technologies are a crucial component for classical cloud computing [Zha18]. They provide the capability to share server clusters as a resource pool and the ability to dynamically allocate virtual resources to customers and applications [Zha18]. It allows the cloud provider to share extensive hardware resources among multiple users through resource partitioning. Despite the simultaneous use of the hardware resource, users are isolated in their virtual environment. Furthermore, cloud providers can aggregate multiple hardware resources into large virtual resources for resource-intensive users. In this way, virtualization enables efficient hardware utilization with simultaneous flexibility. In addition, the virtualization layer creates independence from the underlying hardware. The customer is therefore independent of the underlying physical hardware used by the cloud provider to implement its services. However, these benefits of virtualization in cloud computing come with the overhead cost for virtualization.

3 Problem Statement and Research Objective

This chapter first describes the problem of this thesis and then formulates the research objective. The problem description is divided into a brief outline and a recapitulation of the current situation and the resulting challenges.

3.1 Current Situation

Today, NISQ devices are available in quantum cloud offerings [Cas17]. Manufacturers such as IBM[1] and Rigetti[2] offer access to their QPUs through their cloud platform. Alternatively, third-party cloud services such as Amazon Braket[3] make QPUs available to the public. The term QCaaS describes the new service offered by cloud providers to run a quantum circuit on a QPU hosted in the cloud [LBF+20]. In addition to QPUs, cloud offerings often include classical simulators of QPUs. Both QPUs and simulators can execute quantum circuits, and the term QERs summarizes them. The customer has to select a QER by himself in order to execute a quantum circuit. This decision is not just about choosing between a QPU and a simulator, but often cloud providers offer several QPUs that differ in the number of qubits and their quality, among other things. Thus, the selection influences the execution and its results. Suppose the customer has chosen a QPU. In this case, there are generally two different ways to access the selected QPU: Either the customer reserves an exclusive time slot on a QPU or uses a shared job queue for a public QPU [KTP+20]. In both cases, the quantum cloud provider uses time-based resource sharing to manage access to its QPUs. The only difference is how long a QPU is exclusively assigned to a customer. In the first case, the QPU is made available to the customer exclusively for the entire duration of the reserved time slot; in the second case, only for the execution time of the single quantum circuit.

3.2 Challenges

The way current QCaaS offerings allocate QPUs to a customer and how they divide them between multiple clients leads to shortcomings. The following describes the limitations and identifies three challenges.

Due to the exclusive allocation of the QPU in time-based resource sharing, the utilization of the QPU depends on the utilization of the individual customer. Even a shared job queue does not guarantee high utilization of the QPU's qubits because the execution of a quantum circuit always

[1] https://quantum-computing.ibm.com/

[2] https://www.rigetti.com/

[3] https://aws.amazon.com/braket/

blocks the entire QPU. Qubits of the QPU that are not needed for the execution of the quantum circuit are unused for the entire execution time [DL20; DTNQ19]. These qubits are inactive and cannot be used for other purposes. The first challenge formulates this shortcoming.

Challenge 1
In time-based resource sharing, a QPU is assigned exclusively to one customer. The execution of the customer's quantum circuit blocks the entire QPU and results in unused qubits when executing smaller quantum circuits.

Moreover, the available physical QPUs have a limited number of qubits, which limits the width of quantum circuits that can be executed. The QPU's qubits are an upper bound on the quantum circuit's qubits. Therefore, QPUs that have fewer qubits than the quantum circuit cannot execute it. In QCaaS offerings, the computation of a quantum circuit is realized by running it on a QPU. The time-based resource-sharing model assigns each computational task exclusively to a single QER that performs it. Thus, the capability of a QPU to execute the quantum circuit limits the ability to compute a result since the implementation of the QCaaS model does not allow small QPUs to contribute in any way to the computation of large quantum circuits [PHOW20]. It only allows the computation of quantum circuits that fit into one execution on a single QPU. The second challenge establishes this impairment.

Challenge 2
A QPU can only execute quantum circuits that require at most the QPU's number of qubits. In current QCaaS offerings, small QPUs cannot contribute in any way to the computation of larger quantum circuits.

In addition, the design decision to require customers to manually select the specific physical QPU for their execution has implications for the user and the cloud provider. It is beneficial for researchers who want to have complete control over the selection of physical resources. However, in order to integrate the computation of a quantum circuit, software developers have to deal with resource selection and cannot focus on the pure computation of quantum circuits and their results. They need to identify the QPUs and check their availability. Moreover, developers have to acquire in-depth knowledge about the different QPUs and their physical peculiarities and limitations. It puts the burden of optimal resource selection and its complexity on the customer, even if they do not care about the specific underlying execution hardware.

Furthermore, the fact that customers directly select physical QPUs also has disadvantages for the cloud providers. They have limited control over the usage and utilization of their QPUs. They are dependent on the customer's decision, which may be arbitrary. Even if the users do not care about the specific QPUs, the resource selection depends on them, and thus the resource optimization of the cloud provider depends on the customer. This dependency may prevent the cloud provider from achieving its goal of optimizing resource utilization.

Challenge 3
Customers of QCaaS offerings must manually select the physical QPU. This has implications for both sides:

1. *The customer needs knowledge about the characteristics and availability of the QPUs.*
2. *The QCaaS provider has limited ability to influence the customer's QPU selection and therefore has limited ability to optimize the utilization of physical resources.*

3.3 Research Objective

This section briefly summarizes the challenges presented earlier and formulates the research objective. The QCaaS model combined with time-based resource sharing creates a tight coupling between the computation of a quantum circuit to a particular physical QPU. There is a one-to-one relationship between the computation of a quantum circuit and its execution on a particular QPU. The execution of the quantum circuit blocks the entire resources of the QPU (cf. Challenge 1) and, vice versa, a QPU must perform all the execution of the quantum circuit (cf. Challenge 2). Furthermore, the customer is responsible for the selection of a suitable QPU (cf. Challenge 3). The selection task adds complexity for the customer and makes it difficult for the cloud provider to optimize the QPU selection. Challenge 1 and Challenge 2 refer to a single QPU, while Challenge 3 refers to a pool of QPUs.

The main objective of this thesis is to decouple the computation of quantum circuits from the underlying execution hardware. Therefore, the feasibility and effectiveness of techniques for a layer that abstracts from actual quantum resources to a virtual execution environment are investigated. Of particular interest are techniques that enable partitioning (cf. Challenge 1) and aggregation (cf. Challenge 2) of resources. It also explores how a virtual execution environment can manage and automate the entire execution process, including resource selection, and thus help to reduce the complexity of the execution process and optimize the QER usage of the cloud provider (cf. Challenge 3).

4 Related Work

No current work addresses all Challenges 1 to 3 presented in the previous chapter. However, there are several existing approaches that address each of the challenges. This chapter presents the related work for each challenge.

4.1 Challenge One

Advances in a technique called multiprogramming for quantum computing addresses Challenge 1. The basic idea is to partition a QPU to execute several smaller quantum circuits simultaneously. Tannu and Qureshi [TQ19] performed a case study for a scenario where a quantum circuit requires half or fewer qubits than what is physically available. They investigated whether it is useful to run two copies of this quantum circuit in parallel on a NISQ device to allow more shots per unit time. They observed some scenarios where running two copies in parallel is advantageous. Subsequently, Das et al. [DTNQ19] took this idea of partitioning a QPU and extended it to run two different quantum circuits of the same or different sizes simultaneously on one NISQ machine. They coined the term multiprogramming quantum computing and demonstrated an improvement in the throughput and utilization of the QPU with their approach. Dou and Liu [DL20] published an improved multiprogramming method that outperforms the previous method in terms of result quality and compilation overhead. They have implemented it in a system called QuCloud, which incorporates a new qubit mapping approach and a compilation task scheduler for multiprogramming quantum computers in the cloud [LD21]. Niu and Todri-Sanial [NT21] address the QPU partitioning problem with a hardware-aware multiprogramming compiler that takes into account the hardware topology, calibration data, and crosstalk effect to reliably assign partitions to different quantum circuits. While these works have considered the problem from the perspective of the QPU and its partitioning, in the following, this thesis focuses on quantum circuits and how to aggregate multiple circuits to run on one QPU. These are two different sides of the same problem.

4.2 Challenge Two

Recent work in the area of quantum circuit partitioning addresses Challenge 2. The basic idea is to decompose large quantum circuits into smaller subcircuits, which are then either executed on a QPU or a classical simulator. The method then reconstructs the result of the initial quantum circuit from the measurement results of the subcircuits. In 2018, Suchara et al. [SAC+] introduced this concept of quantum circuit partitioning in a hybrid quantum-classical architecture. While their work gives an overview of what an architecture based on quantum circuit partitioning might look like, it lacks the technical details of how circuit partitioning and recombination are performed. The following papers present techniques for circuit partitioning.

Peng et al. [PHOW20] developed a systematic approach to partitioning quantum circuits. Their method is based on cutting an identity gate. There are two types of resulting subcircuits. The first type measures the qubit in the Pauli base at the identity gate position, the second initializes the qubit at the cut again. Their work includes a protocol that reconstructs the actual result from the results of the subcircuits. This approach generates an overhead in time of about $2^{O(K)}$ for K qubits of inter-subcircuit quantum communication compared to the execution of the initial quantum circuit. Mitarai and Fujii [MF19] call the cutting method of Peng et al. [PHOW20] a "time-like" partition since this approach cuts a quantum circuit's wire. In contrast, they call their cutting technique "space-like" because it decomposes controlled gates into a sequence of single-qubit operations that decompose a quantum circuit into subcircuits.

In addition to the theoretical cutting frameworks, Ayral et al. [ALS+20] and Perlin et al. [PTP+19] published the first implementations and demonstrated the feasibility of the cutting methods. They show that it is possible to perform computations for quantum circuit sizes exceeding the size of the QPU and to obtain better success probabilities for smaller circuit sizes. In the following, Tang et al. [TTS+21] developed CutQC, a more sophisticated cutting tool. It is the first tool that automates the identification of cuts. This automation is implemented by a mixed-integer programming cut-searcher that finds optimal cuts given any quantum circuit as input. Quantum devices then evaluate different variants of the small subcircuits that result from the cuts. In a final step, the reconstructor reproduces the probability distributions of the original circuit.

4.3 Challenge Three

Currently, there is no work that addresses all aspects of Challenge 3 in quantum computing. It involves automated and optimized mapping of quantum circuit computation to executions of physical resources, including aggregation and partitioning of quantum circuits. However, initial steps have been taken to automate the hardware selection process. Suchara et al. [SKF+13] introduced the Quantum Resource Estimator, which estimates quantities such as the number of physical qubits, the execution time, and the success probability of computing a quantum algorithm on a QPU. This assessment can help with the QPU selection. Salm et al. [SBB+20] developed the concept of a NISQ analyzer that automates the analysis and selection of implementations of quantum algorithms and suitable quantum computers that can run the selected implementation with a certain input data. Although these works are crucial steps toward automating hardware selection, they work with abstract algorithms as input rather than a concrete implementation of an algorithm as a quantum circuit. Furthermore, no work addresses the systematic combination of quantum circuit aggregation and partitioning in a single system.

Nevertheless, in classical cloud computing, the discovery, selection, allocation, and mapping of physical resources are solved. Cloud providers are responsible for mapping user workloads to physical resources based on quality-of-service requirements. It is called resource management and includes resource provisioning, resource scheduling, and resource monitoring [SC16]. Resource management refers to predicting the number of resources that are best suited for each workload and enables cloud providers to consolidate workloads while meeting service level agreements and quality-of-service requirements [WBW15]. It has two main purposes: enable the execution of tasks and optimize infrastructural efficiency based on a set of specified objectives [GCM17].

5 Concept

In this chapter, the concept of the thesis is presented to solve the problem introduced in Chapter 3. The proposed solution adapts concepts from the field of virtualization to quantum computing. It separates the logical computation of a quantum circuit from the underlying executing QERs. An intermediate software layer abstracts from the physical QERs and combines them into a unified and hardware-independent execution interface, the virtual execution environment. Figure 5.1 shows the virtual execution environment as an intermediary between the logical computation request of a quantum circuit and the executing QERs.

The virtual execution environment receives a quantum circuit as input and outputs the execution result. To produce the results, it utilizes the underlying QERs. It encapsulates the physical constraints and idiosyncrasies of the single QERs. Moreover, the virtual execution environment maps the workload of quantum circuit computations to QERs without user interaction. In this process, there is no one-to-one relationship between the computation of a quantum circuit and a single QER. It uses quantum circuit aggregation and partitioning to break this relationship.

Quantum circuit aggregation combines the computation of multiple quantum circuits into one circuit instance. Thus, a QER running the aggregate executes all initial quantum circuits simultaneously. It partitions the resource with respect to its qubits between the aggregated quantum circuits. Aggregating quantum circuits that require fewer qubits than the QER provides allows for more efficient qubit usage and higher throughput on the QERs. This is a solution for Challenge 1.

Quantum circuit partitioning allows the workload of computing a large quantum circuit to be divided into the execution of smaller subcircuits. Thus, QERs with fewer qubits can participate in the computation of large circuit instances. The subcircuit execution can be distributed among several different QERs, or one QER can execute all subcircuits in sequence. In both cases, quantum circuit partitioning pools resources to compute the quantum circuit: either the resources of multiple QERs

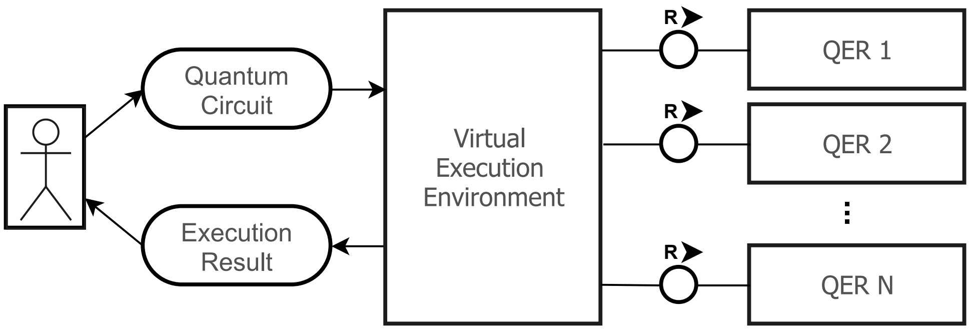


Figure 5.1: A client executes a quantum circuit in the virtual execution environment. The virtual execution environment uses one or multiple of the underlying quantum execution resources to execute the quantum circuit.

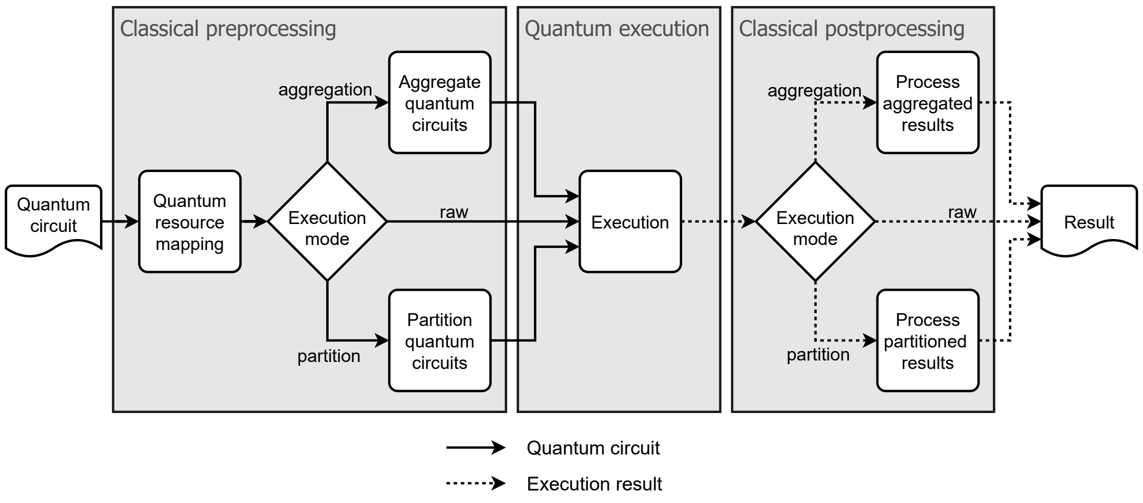


Figure 5.2: Conceptual virtualization process flow chart. Rectangles with rounded corners represent processing steps, diamonds represent decisions, and rectangles with a wavy side represent data.

or the capacity of one QER over time. Thus, it enables the aggregation of resources and deals with Challenge 2. In the remainder of the thesis, the terms aggregation and partitioning always refer to the aggregation and partitioning of quantum circuits instead of hardware resources.

The virtual execution environment overcomes Challenge 3 by automating the mapping between the computation of a quantum circuit to appropriate QERs. The mapping process can follow different optimization strategies that govern the aggregation and partitioning of quantum circuits. The remainder of the chapter explains how the virtual execution environment works conceptually. The virtualization process that a computation request goes through is presented, and the critical parts of the system are described in detail.

5.1 Virtualization Process

The virtualization process describes how the system maps the logical computation of a quantum circuit to the physical execution of QERs and how it produces the corresponding execution result. The process is divided into three major parts: classical preprocessing, execution on QERs, and classical postprocessing. The input of the process is a quantum circuit. The quantum circuit goes through the preprocessing step that maps the computation workload to QERs. The next step is to execute the preprocessing outcome on the QERs. A classical postprocessing routine is then applied to the execution results. The output of the virtualization process is the corresponding logical result to the input quantum circuit. Figure 5.2 shows an overview of the conceptual process in the virtual execution environment.

Classical preprocessing begins with quantum resource mapping. Therefore, it analyzes the quantum circuit and checks the available QERs. Then, the system must make two landmark decisions to map the workload of computing the quantum circuit to executions on QERs. It must select QERs and an execution mode for further processing of the quantum circuit. The execution mode is either aggregating the quantum circuit with another circuit, partitioning the quantum circuit into subcircuits, or leaving the quantum circuit as it is, which is called raw execution. The decisions are based on the properties of the QERs and the quantum circuit. Depending on the execution mode selected, the system applies further processing to the quantum circuit. In the case of aggregated or partitioned execution, the appropriate processing steps are applied. The output of the preprocessing is quantum circuits suitable for the execution on the specific QERs.

The next step is execution. The virtualization process passes the quantum circuits resulting from the preprocessing to the specified QERs. It handles the interaction with the QERs and retrieves the results when they are available.

Finally, the postprocessing takes place. The process decides whether additional postprocessing is required and applies it to the aggregated and partitioned quantum circuits. Aggregation postprocessing extracts the results of the combined circuits. Partition postprocessing collects the results until all necessary information is available and then calculates the overall result. The output of the postprocessing is the result of the original input quantum circuit.

The described virtualization process supports the simultaneous execution of subcircuits of a partitioned quantum circuit on multiple QERs. However, to simplify the quantum resource mapping and the following processing steps, this case is not considered in the detailed elaboration of the concept. The following concept description focuses on the successive execution of the subcircuits on one QER when partitioning a quantum circuit.

5.2 Quantum Resource Mapping

Quantum resource mapping is the first step of classical preprocessing. It is crucial for all the following steps. The mapping determines the execution mode of the quantum circuit and the executing QER. There is no straightforward solution to these two decisions. Instead, there are different strategies for selecting QER and execution mode. The suitability of a strategy depends on the optimization goal. Possible optimization goals are, for example, the quality of the execution results, the utilization of the QERs, or the waiting time for an execution result.

Depending on the optimization goal, different properties of the quantum circuit and the QER are relevant. Important properties of quantum circuits can be the width, depth, gates used, and the connectivity between qubits. Substantial properties of the QER may be its availability, the number of qubits, current workload, whether it is a QPU or a simulator, supported base gates, and error rates.

The execution mode and the QER must match since the resulting quantum circuits of the selected execution mode must be executed on the selected QER. So the choice of one affects the other. However, the correct selection sequence is not generally obvious and depends on the given optimization goal. The system can first select the QER and then, depending on the selection, the execution mode or vice versa. It first selects the best execution mode for a quantum circuit and then the specific QER.

For example, suppose the primary goal is a low response time under the secondary goal of efficient qubit usage. In this case, the system can first select the QER with the lowest current waiting time and then choose the execution mode based on the number of qubits of the QER and the width of the quantum circuit. There are the following three cases:

1. The number of qubits of the quantum circuit is equal to or slightly smaller than the available QER size. In this case, the quantum circuit does not need to be processed further in order to be executed.
2. The quantum circuit requires more qubits than are available. In this scenario, the quantum circuits must be partitioned into smaller subcircuits.
3. The quantum circuit requires significantly fewer qubits than are available. Multiple quantum circuits from this category can be aggregated into one larger quantum circuit.

5.3 Aggregation

This section covers the basic concept of the aggregation of quantum circuits and how to retrieve the results from the aggregated quantum circuit result. The aggregation procedure is divided into a pre- and postprocessing part, which must be carried out before and after the execution. The concept of the aggregation method is described for the aggregation of two quantum circuits. Nevertheless, this procedure can be extended to more than two circuits by iteratively applying the procedure to two quantum circuits. The concepts are described below.

5.3.1 Preprocessing

The first step in aggregation preprocessing is to delay quantum circuits until two quantum circuits are available for aggregation. These two quantum circuits are then passed as input to the aggregation method. The output is a quantum circuit containing the qubits and the operation of both quantum circuits. In addition to the aggregated circuit, the method also generates a mapping between the input circuit qubits and the aggregated circuit qubits to reconstruct the results from the aggregated result.

Formally, the aggregation works as follows. The two input quantum circuits are described by $C_1 = U_1 |\psi\rangle$ and $C_2 = U_2 |\phi\rangle$, respectively, where $|\psi\rangle$ is the state of an n qubit register and $|\phi\rangle$ is the state of an m qubit register. The aggregate circuit is then defined by $(U_1 \otimes U_2)(|\psi\rangle \otimes |\phi\rangle)$. It has $n + m$ qubits. Figure 5.3 shows the initial circuits and the resulting aggregated circuit.

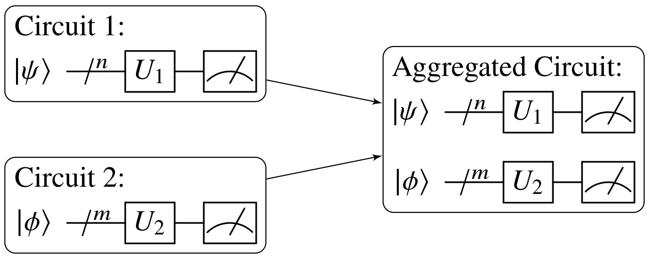


Figure 5.3: Aggregation of two quantum circuits

The generated qubit mapping is a function that maps the qubits for each input quantum circuit to the qubits of the aggregated circuit. Assume that the n qubit register of quantum circuit C_1 has the qubits $Q_1 = \{q_0^1, \dots, q_{n-1}^1\}$, the m qubit register of quantum circuit C_2 has qubits $Q_2 = \{q_0^2, \dots, q_{m-1}^2\}$, and the aggregated quantum circuit register uses qubits $Q_{\text{agg}} = \{q_0, \dots, q_{n+m-1}\}$. A qubit mapping f is a bijective function between the following sets of qubits:

$$f : Q_1 \cup Q_2 \mapsto Q_{\text{agg}} \tag{5.1}$$

5.3.2 Postprocessing

The postprocessing of the aggregation receives as input the execution result of the aggregated circuit and the qubit mapping. It produces two separate results for the two initial quantum circuits. The result of one shot of the aggregated circuit is a bit sequence x of length $n + m$:

$$x = x_{q_{n+m-1}} \dots x_{q_1} x_{q_0} \quad 0 \leq i < m + n : x_{q_i} \in \{0, 1\} \tag{5.2}$$

This shot accounts for one shot in each of the results of the initial circuits C_1 and C_2. Their corresponding result states are obtained by decomposing x into substrings according to the qubit mapping f. The result state for the circuit C_1 is

$$x_1 = x_{f(q_{n-1}^1)} \dots x_{f(q_0^1)} x_{f(q_0^1)} \tag{5.3}$$

and the result state for circuit C_2 is

$$x_2 = x_{f(q_{m-1}^2)} \dots x_{f(q_1^2)} x_{f(q_0^2)}. \tag{5.4}$$

Applying this to all the shots in the aggregated result gives the overall results for the two initial circuits.

5.4 Partitioning

In this section, we introduce the concept of partitioning a quantum circuit into smaller subcircuits. First, the theoretical background is briefly reviewed. Then, the concept of the pre- and postprocessing from the virtualization process is explained.

5.4.1 Background

The technique behind partitioning a quantum circuit is circuit cutting. In the following, the technique of Peng et al. [PHOW20] is described. Their method partitions a quantum circuit by cutting qubit wires representing time from left to right. Therefore, these cuts are called time-wise cuts [TTS+21]. A wire with no operation is equal to a wire on which an identity operation is performed. The key idea is that the identity gate's expected output can be reproduced with a series of measurements and subsequent qubit initializations that separate the gate into two parts.

The mathematical background and proof can be found in the work of Peng et al. [PHOW20]. The following provides a basic understanding of this technique. Since the normalized Pauli matrices $\{I, X, Y, Z\}/\sqrt{2}$ form an orthonormal matrix basis [PHOW20], any 2×2 matrix A can be decomposed by projection onto the Pauli basis as

$$A = \frac{\mathrm{Tr}(AI)I + \mathrm{Tr}(AX)X + \mathrm{Tr}(AY)Y + \mathrm{Tr}(AZ)Z}{2} \tag{5.5}$$

Given that the Pauli matrices are unitary, they can be decomposed into their eigenbasis:

$$I = |0\rangle\langle 0| + |1\rangle\langle 1| \tag{5.6}$$
$$X = |+\rangle\langle +| - |-\rangle\langle -| \tag{5.7}$$
$$Y = |i\rangle\langle i| - |-i\rangle\langle -i| \tag{5.8}$$
$$Z = |0\rangle\langle 0| - |1\rangle\langle 1| \tag{5.9}$$

By applying the eigenbasis decomposition, rearranging the terms and factorizing, we obtain the following equation [TTS+21]:

$$A = \frac{A_1 + A_2 + A_3 + A_4}{2} \tag{5.10}$$

where

$$A_1 = [\mathrm{Tr}(AI) + \mathrm{Tr}(AZ)Z]\,|0\rangle\langle 0| \tag{5.11}$$
$$A_2 = [\mathrm{Tr}(AI) - \mathrm{Tr}(AZ)Z]\,|1\rangle\langle 1| \tag{5.12}$$
$$\begin{aligned} A_3 &= \mathrm{Tr}(AX)\,[|+\rangle\langle +| - |-\rangle\langle -|] \\ &= \mathrm{Tr}(AX)\,[2\,|+\rangle\langle +| - |0\rangle\langle 0| - |1\rangle\langle 1|] \end{aligned} \tag{5.13}$$
$$\begin{aligned} A_4 &= \mathrm{Tr}(AY)\,[|i\rangle\langle i| - |-i\rangle\langle -i|] \\ &= \mathrm{Tr}(AY)\,[2\,|i\rangle\langle i| - |0\rangle\langle 0| - |1\rangle\langle 1|] \end{aligned} \tag{5.14}$$

In A_3 and A_4, the following equations are used to get rid of $|-\rangle\langle -|$ and $|-i\rangle\langle -i|$, respectively, and replace them with terms already in use.

$$|-\rangle\langle -| = -\,|+\rangle\langle +| + |0\rangle\langle 0| + |1\rangle\langle 1| \tag{5.15}$$
$$|-i\rangle\langle -i| = -\,|i\rangle\langle i| + |0\rangle\langle 0| + |1\rangle\langle 1| \tag{5.16}$$

In the context of the density operator formulations (cf. Section 2.1.5), the trace operator represents the measurement's expected value in one of the Pauli bases. Repeated measurements converge to the expected value. Thus, a trace operator physically corresponds to the measurement of the qubit. Each of the density matrices physically corresponds to the initialization of the qubit in one of the eigenstates [TTS+21]. Since measuring a qubit in either the I or Z basis physically yields the same results, we need to evaluate seven different subcircuits: three for the measurements in the Pauli bases I, X, and Y, and four subcircuits with the qubit initialization for each of the eigenstates $|0\rangle$, $|1\rangle$, $|+\rangle$, and $|i\rangle$. Figure 5.4 shows the subcircuits and how the original output of the circuit can be reconstructed by summing over the tensor products of the different subcircuits.

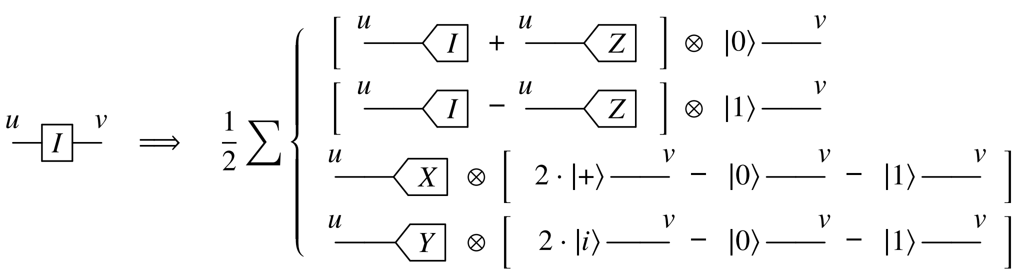


Figure 5.4: Method of cutting one qubit wire. On the left side is the identity gate between u and v in a quantum circuit. The right side shows the resulting subcircuits and the reconstruction process involved in cutting the identity gate. Adapted from Peng et al. [PHOW20].

5.4.2 Preprocessing

Partition preprocessing receives one quantum circuit as input and generates multiple quantum circuits as output. To determine where to cut the quantum circuit, Tang et al. [TTS+21] present a mixed-integer program that automatically identifies cuts that require the least amount of classical postprocessing. For a given cut position, the preprocessing procedure generates the subcircuits with the required measurements and qubit initializations for each cut. The number of subcircuits created depends on the number of cuts required to divide the input circuit into separate subcircuits. Figure 5.5 shows an example cut of an $(n + m + 1)$-qubit circuit where only one cut is needed for separation. The output at the cut of Subcircuit 1 must be measured in the Pauli basis, that is $M \in \{I, X, Y\}$. The input at the cut of Subcircuit 2 must be initialized with all values $Q \in \{|0\rangle, |1\rangle, |+\rangle, |i\rangle\}$. Therefore, three different versions of Subcircuit 1 and four versions of Subcircuit 2 must be executed. If multiple cuts are required to partition the quantum circuit, the cutting procedure must be applied iteratively to the resulting subcircuits. In addition to the subcircuits, the preprocessing generates an index with all subcircuits, which is used for the subsequent reconstruction of the results.

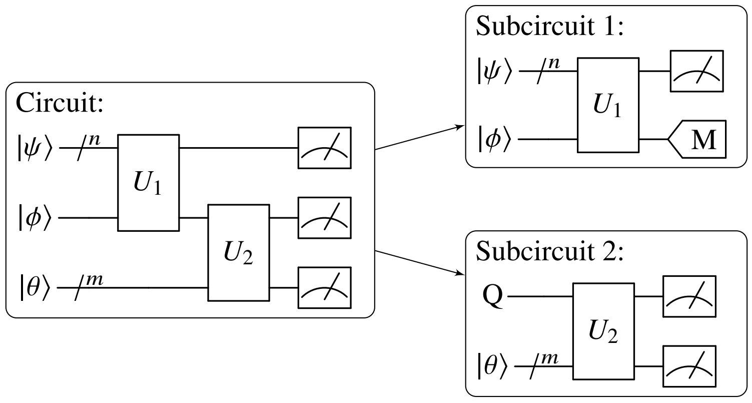


Figure 5.5: Example of cutting an $(n + m + 1)$-qubit circuit into two smaller subcircuits of $n + 1$ and $m + 1$ qubits. $Q \in \{|0\rangle, |1\rangle, |+\rangle, |i\rangle\}$ and $M \in \{I, X, Y\}$. Adapted from Tang et al. [TTS+21].

5.4.3 Postprocessing

The postprocessing procedure collects the results of the subcircuits. Using the previously generated subcircuit index, it checks the availability of all results. When the subcircuit results are complete, it reconstructs the expected result of the original circuit. The postprocessing procedure applies the reconstruction method to each cut as described in Figure 5.4. For this purpose, the subcircuit index codes how the individual results are to be combined.

6 Implementation

In the previous chapter, the concept of the virtual execution environment was introduced. This chapter describes a prototypical Python implementation of it. The main task is to implement the virtualization process, but also an interface to communicate with and integrate the virtual execution environment must be implemented.

The implemented system accesses QPUs via a Quantum Computing Service Platform (QCSP). To this end, the implementation uses IBM's QCSP, called IBM Quantum Experience (IBM QX). IBM QX is a cloud platform that provides access to IBM's QERs. It offers QPUs in a QCaaS model with up to 15 qubits and a simulator with 32 qubits in the free version. The implementation uses the open-source software development kit Qiskit by Abraham et al. [AAA+19] to work with quantum computers and develop quantum applications. Qiskit provides OpenQASM as a quantum assembly language. QpenQASM is human-readable and has elements of C and assembly languages [CBSG17]. It allows a programmatic description of quantum circuits. Furthermore, the implementation uses parts of the functionality of the framework CutQC [TTS+20]. Tang et al. [TTS+21] developed the CutQC framework to evaluate large quantum circuits with small quantum devices. Therefore, CutQC uses a partitioning of the large quantum circuits into smaller subcircuits.

This chapter is structured as follows. First, the architecture of the system is presented. This is followed by a detailed description of the implementation of the virtualization processes. Finally, the API interface is described.

6.1 Architecture

A pipes-and-filter architecture realizes the implementation of the virtual execution environment. A filter is a processing step and has at least one data input and one data output. Pipes describe the data flow and connect filters. In the preprocessing of the virtual execution environment, filters transform quantum circuits, and in postprocessing, filters reconstruct the results. For quantum circuit execution, the virtual execution environment uses IBM QX to access QERs. The virtual execution environment is integrated with the virtualization service, making the virtual execution environment remotely accessible via a REST API. The virtualization service allows interaction with multiple clients and can serve them simultaneously.

Figure 6.1 shows the entire system, including the clients, the virtualization service that consists of the virtual execution environment and the API, and the QCSP. It shows their components and how they relate to each other. A client sends a request to the virtualization layer via an HTTP request to the REST API. The generated request contains the quantum circuits for execution and optional configuration data. The API processes the request, creates a task, and stores the task in a database. The client can query the task and its status from the database and initiate the execution. After that, the virtualization process starts as described in Chapter 5. The Quantum Execution

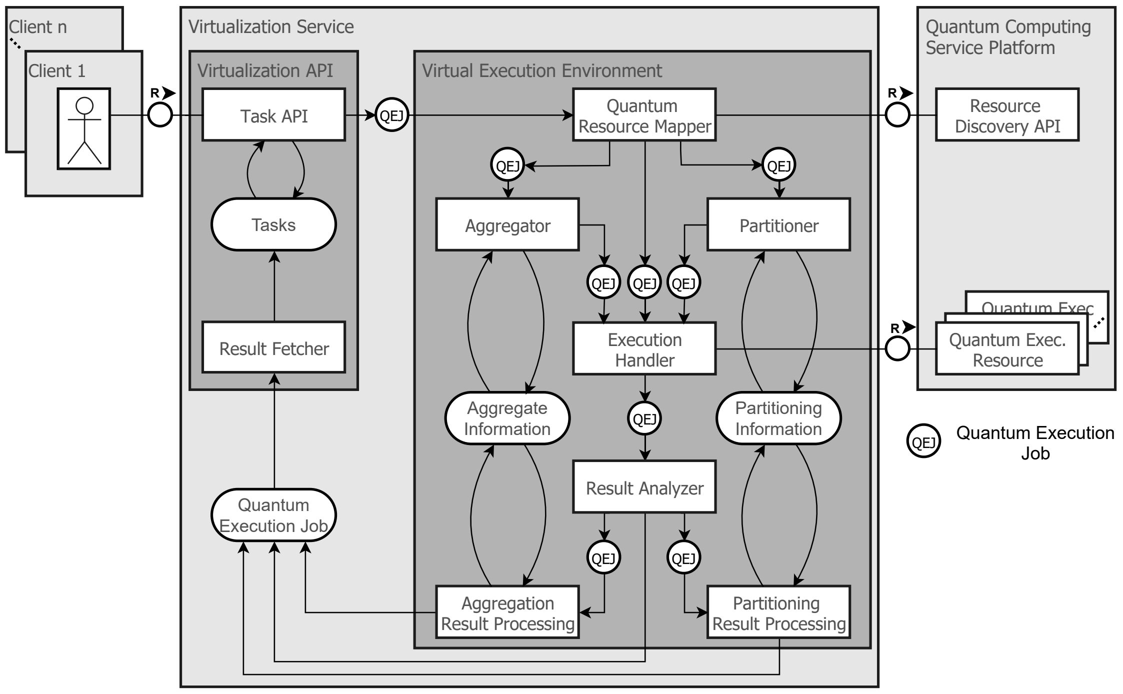


Figure 6.1: Architecture as FMC block diagram

Mapper has to decide how to further process the quantum circuit. The decision consists of two parts: which QER should be used and whether the quantum circuit should be executed as aggregated, partitioned, or raw. Therefore, the Quantum Execution Mapper incorporates data about the available QERs, which it obtains by making a request to the QCSP. Based on the decision by the Quantum Execution Mapper, the quantum circuits are forwarded to the Aggregator, Partitioner, or directly to the Execution Handler. When processing the circuits, the Aggregator and the Partitioner store data to reconstruct the actual results later.

After the preprocessing is complete, the quantum circuits are passed to the Execution Handler. The Execution Handler manages the communication with the QCSP, executes the quantum circuits on the selected QER, and returns the results. Then the postprocessing of the retrieved results begins. The first step is for the Result Analyzer to decide whether postprocessing is required, and if so, what type of postprocessing is required, and to forward the results. Aggregation Result Processing and the Partition Result Processing reconstruct the original result. Therefore, they use the information previously stored by the Aggregator and Partitioner. The final results of the initial request to the virtual execution environment are now available. The Result Fetcher collects the results and matches them with the initial request. The client can now retrieve the result via a request to the API.

In Figure 6.1 the error-handling has been omitted in favor of the readability of the graphic. However, there is an error queue that receives all errors that occur in the system. Each component can place jobs affected by an error in the error queue. The Result Fetcher reads not only the successful results but also the error queue and updates failed tasks.

6.2 Virtual Execution Environment

The virtual execution environment is the core component of the system. It implements the virtualization process described in Chapter 5. The virtual execution environment receives as input a quantum circuit with optional configuration wrapped in a Quantum Execution Job. It processes the Quantum Execution Job and outputs the execution results as an associated probability distribution. The virtualization service allows the virtual execution environment to be used as a service through its REST API. However, a developer can also import the virtual execution environment locally in Python and use it in combination with Qiskit.

6.2.1 Quantum Execution Job

A Quantum Execution Job is the internal data structure used for communication between the various components. Each Quantum Execution Job has at least the following data fields:

- a unique *ID* to identify the job
- a *quantum circuit* for the execution
- the number of *shots*
- an *execution type*
- a job-related *configuration*
- a *result*

The execution type indicates whether the circuit went through the aggregation or partitioning routine. At the beginning, the execution type is always *raw*. When the Quantum Resource Mapping changes the execution type to *aggregation* or *partitioning*, the Quantum Execution Job itself and all jobs that emerge from it through aggregation and partitioning retain the respective execution type. The default configuration is empty. If it contains a key-value pair, it overrides the system's default configuration value for the specified key for that particular Quantum Execution Job. After the execution of the quantum circuit, the Quantum Execution Job is extended with the corresponding probability distribution of the result. As long as there is no result, the result data field is empty.

6.2.2 Quantum Resource Mapper

The Quantum Resource Mapper is a core component of the virtual execution environment. It receives as input a Quantum Execution Job and determines the mapping of the computation of its quantum circuit to a QER. Therefore, it has to choose the QER for the execution and the execution mode of the quantum circuit.

To choose the QER, the Quantum Resource Mapper discovers accessible QERs of the QCSP and retrieves their properties, such as the number of qubits and whether it is a simulator or a QPU. It periodically queries the availability and utilization of the QERs and maintains an internal knowledge base that is used for the resource mapping. To approximate the utilization of a QER, the Quantum Resource Mapper uses the number of jobs in the IBM QX public queues.

Moreover, the Quantum Resource Mapper's behavior can be customized via a configuration file and the optional configuration included with the Quantum Execution Job. The accessible QERs can be restricted by a white or black list. Furthermore, it allows restricting the use of QERs via requirements for their properties. Currently implemented constraints are minimum and maximum values for the number of qubits and the quantum volume of a QER and whether simulators are allowed. In addition, it is possible to allow only certain execution types. However, in the default configuration, the Quantum Resource Mapper can choose from all three execution types: aggregated, partitioned, and raw.

Based on the Quantum Execution Job, the cached information about the QERs, and the configuration, the Quantum Resource Mapper outputs an execution type and a QER. Two exemplary strategies are implemented for this purpose, which can be selected via the configuration.

Strategy: High throughput

The goal of this strategy is to maximize throughput. Therefore, it prefers aggregation execution mode over raw, and raw over partitioning execution mode because aggregation executes multiple quantum circuits in one run, raw executes only one, and partitioning requires multiple runs on a QER for one quantum circuit. To select the QER, the Quantum Resource Mapper first searches for available QERs that can execute the quantum circuit in aggregate mode. These are QERs that have at least twice the number of qubits required by the quantum circuit. If no available QER provides this number of qubits, the Quantum Resource Mapper tries raw execution mode and then the partitioned mode. If multiple QERs are possible, the Quantum Resource Mapper uses the QER's utilization as a tie-breaker and selects the one that is less utilized.

Strategy: Low waiting time

This strategy aims to reduce the waiting time until the execution is completed. It first selects the QER. Therefore, it chooses the least busy operational QER with enough qubits. The least busy QER is the one with the fewest waiting jobs. If no QER with enough qubits is available, the strategy chooses the least busy QER with the most qubits. Now the Quantum Resource Mapper can select the execution type of the quantum circuit. Thus, it takes into account the number of qubits of the selected QER and quantum circuit. If the number of qubits of the quantum circuit exceeds the QER qubits, the circuit must be partitioned. If the number of qubits of the quantum circuit is not more than half of the quantum resource qubits, the circuit can be aggregated with another circuit. Otherwise, no further processing is required, and the Quantum Resource Mapper can pass the quantum circuit directly to the Execution Handler.

The low-waiting-time strategy favors QERs that are able to execute the quantum circuit without partitioning since partitioning itself is a computationally intensive and time-consuming task. Moreover, one quantum circuit leads to the execution of multiple subcircuits, which increases the waiting time.

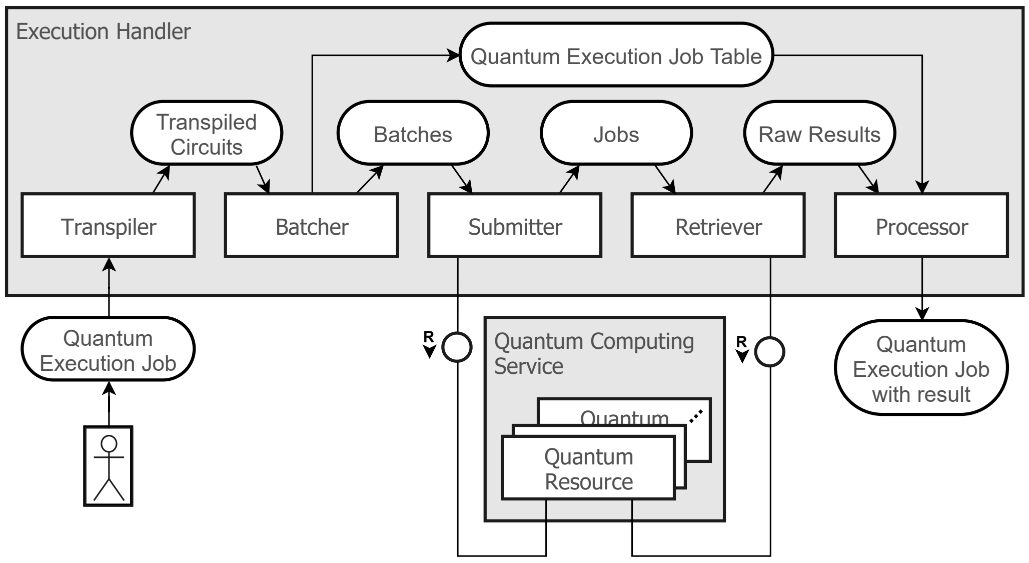


Figure 6.2: Execution Handler as FMC block diagram

6.2.3 Execution Handler

The Execution Handler takes care of the execution of a quantum circuit on a QER. It receives as input a Quantum Execution Job containing the quantum circuit, the QER's name on which the quantum circuit is to be executed, and the number of shots. It does not matter how many shots the QER physically supports in one round of execution since the Execution Handler can split a logical execution with many shots into several physical execution rounds. Ultimately, the Execution Handler outputs the quantum circuit's execution result for the specified QER and shots.

The Execution Handler aims to enable the execution of quantum circuits as efficiently as possible and with the highest possible throughput. The limiting factor for the system's execution capacity is the limited number of execution jobs on IBM QX. IBM QX allows only five pending execution jobs per QER. The number of quantum circuits that can be included in one job varies between the different QERs. However, most of the QERs have a maximum batch size of 75 quantum circuits per job. Thus, the execution must exploit the maximum quantum circuit limit per job to achieve high throughput and low waiting time. To realize this, the Execution Handler consists of five subcomponents: the Transpiler, the Batcher, the Submitter, the Retriever, and the Processor. Figure 6.2 shows the subcomponents and their communication via data.

Transpiler

The Transpiler consists of two subroutines. The first maintains a list per QER of quantum circuits that need to be transpiled. It receives the quantum circuit and places it in the appropriate list. If the list size exceeds the maximum batch size for the QER or a timeout occurs, the quantum circuits are marked for transpilation. The next subroutine of the Transpiler is the actual transpiling process. The Transpiler fetches the lists of marked quantum circuits and transpiles the quantum circuits one by one for the specified QER. It processes the quantum circuit in each list in parallel and uses multiple

CPU cores, if available. The Transpiler is placed at the first position in the Execution Handler in order to use the waiting time of the Batcher during batch creation in the next step for transpiling the quantum circuits.

Batcher

The next step in the Execution Handler is the Batcher. Its job is to bundle the quantum circuits into batches per QER most efficiently for the batch size limit of one job on the QER. Therefore, the Batcher delays further processing of transpiled quantum circuits until the batch size limit is reached or a timeout occurs. Moreover, the Batcher abstracts from the shot limit of the QER. The QERs of IBM QX have a shot limit of 8192 shots per execution of a quantum circuit. However, assume that the specified number of shots in the Quantum Execution Job exceeds the maximum number of shots for the QER. In this case, the batcher creates multiple instances of the quantum circuit such that the sum of the shots of the instances is equal to the specified number of shots. Until now, the Quantum Execution Job has been included in every step. Now the Batcher separates the Quantum Execution Job from its quantum circuits in the batch and stores a mapping between them in the Quantum Execution Job Table.

Submitter

The Submitter's task is to start the execution of the previously created batches. Therefore, it takes the batches and sends executables to the QERs of the QCSP. If there is a limit of active submitted jobs on a QER waiting to be executed, the Submitter will postpone the execution of jobs until it is possible to do so. The Submitter passes a handler of the submitted job to the Retriever.

Retriever

The Retriever receives the submitted jobs and periodically checks their condition. When a job is in a final state, the raw result of the job is retrieved. The correct order of the executed jobs is essential in the Execution Handler to reconstruct the results of quantum circuits that have been split into multiple instances distributed over multiple batches. In rare cases, IBM QX perturbs the order of jobs. If multiple jobs are added to a QER in rapid succession, the validation of a subsequent job can be completed sooner. Then the execution of the jobs gets out of sequence. However, the Retriever guarantees that it passes the results to the Processor in the same order in which the batches were created for the QER.

Processor

The last step in the Execution Handler is the Processor. The Processor reconstructs the results of the original Quantum Execution Job. Therefore, it matches the corresponding quantum circuit executions with the Quantum Execution Job using the information in the Quantum Execution Job Table. When all executions of a Quantum Execution Job are complete, it starts their processing. The main task of the Processor is to combine the results of multiple quantum circuits generated by the Batcher to bypass the shot limit into a single result. For this purpose, the Processor sums the shots of these circuits' results. It attaches the single end result to the Quantum Execution Job, which is then output.

6.2.4 Aggregator

The Aggregator implements the aggregation routine described in Section 5.3. It aggregates the circuits of two consecutive Quantum Execution Jobs for the same QER. So when the first Quantum Execution Job arrives for a QER, it stores it and starts a timer. The Aggregator waits until a second Quantum Execution Job is available for the same QER or a timeout occurs. In the event of a timeout, the pending Quantum Execution Job is passed to the Execution Handler for raw execution without aggregation. If a second Quantum Execution Job is available, the Aggregator takes those jobs and starts the aggregation routine. The routine creates a new empty quantum circuit. The qubits of the new circuit are the sum of the qubits of the input quantum circuits. The same applies to the new circuit's classic bits for measurement. Then the aggregation routine adds all the operations of the two quantum circuits to the aggregated quantum circuit. At the end, a new Quantum Execution Job is created containing the aggregated quantum circuit. The Aggregator passes the new Quantum Execution Job representing the aggregation to the Execution Handler. It also stores two mappings. The first mapping is between the aggregated Quantum Execution Job and the two initial Quantum Execution Jobs to remember their relationship. The second mapping is from the initial circuits' qubits to the aggregated circuit's qubits to extract the results of the initial quantum jobs from the aggregated result.

6.2.5 Aggregation Result Processing

Aggregation Result Processing receives the aggregated result of the Quantum Execution Job. To reconstruct the two results of the initial quantum circuits, it retrieves the stored qubit mapping generated by the Aggregator. Then it starts the postprocessing routine as described in Section 5.3. For each state in the aggregated result, postprocessing identifies the corresponding result states of the initial quantum circuits and adds the measured count values. When finished, the method loads the original Quantum Execution Jobs and appends their results.

6.2.6 Partitioner

The Partitioner implements the circuit cutting procedure described in Section 5.4. It receives as input a target QER for execution and a Quantum Execution Job. The Partitioner outputs several subcircuits that are executable on the given QER. To accomplish this, the Partitioner executes two subroutines from the CutQC [TTS+20] framework. The first is the optimal cut search. It is coded as a Mixed-Integer Program and solved with the Gurobi[1] solver. Tang et al. [TTS+21] give a detailed description of the Mixed-Integer Program for cut search in their work. The cut searcher offers the possibility to set the maximum number of cuts, the maximum number of resulting parts, and the maximum width of the subcircuits via parameters. The maximum number of cuts and resulting parts

[1] https://www.gurobi.com/

are specified in the configuration file and can be overridden for a particular Quantum Execution Job via its own attached configuration. By default, the maximum width of the generated subcircuits is limited by the number of qubits of the QER. However, it is also possible to set a maximum subcircuit size in the Quantum Execution Job configuration. Then, the minimum of the two values for the maximum subcircuit width is used to ensure that they are executable on the QER. Nevertheless, there may be no feasible solution to cut the given quantum circuit for the chosen QER with the given parameters. In this case, the Partitioner passes the quantum circuit to error handling. Otherwise, in the next step, the Partitioner generates all the subcircuits for the various measurements and qubit initializations using the previously calculated cut solution. The Partitioner creates a new Quantum Execution Job for each subcircuit and submits these jobs to the Execution Handler. It also stores the cut solution and other information about the generated subcircuits to reconstruct the result later as well as a mapping between the initial and the partitioned Quantum Execution Jobs.

6.2.7 Partition Result Processing

The Partition Result Processing implements the circuit-cutting reconstruction procedure as described in Section 5.4. It receives as input the subcircuit's results and recreates as output the original circuit's result using the previously stored information about the cut. The Partition Result Processing consists of two parts. The first component receives the subcircuits results and writes them to files. It matches the subcircuits to the initial Quantum Execution Job. As soon as all subcircuit results are available, it notifies the second component to start postprocessing the results. The Partition Result Processing invokes the postprocessing step of the CutQC framework [TTS+20]. The procedure, implemented in C, reads in all previously stored files and reconstructs the original quantum circuit's probability distribution. For this purpose, it uses the Intel oneAPI Math Kernel Library[2]. When the procedure is complete, the result is appended to the original Quantum Execution Job and any files created for postprocessing are deleted.

6.3 Virtualization API

The virtualization API manages the interaction between the clients and the virtual execution environment. It offers a REST API for clients to send quantum circuits, execute them through the virtual execution environment, and retrieve their results. The API is implemented with the Python module Flask[3]. Flask is a lightweight framework for developing web applications. As a database for managing and storing the API data, the system uses a MongoDB[4] database in the background. Since the execution of a quantum circuit can take time, the API is implemented in such a way that long-running tasks can be executed asynchronously. The design of the API is inspired by the Asynchronous Request-Reply pattern[5]. A client does not have to wait synchronously until the execution of the quantum circuit is complete. Instead, the client uses multiple synchronous requests. The first request passes the quantum circuit to the virtualization service and creates a

[2]https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html
[3]https://github.com/pallets/flask
[4]https://www.mongodb.com/
[5]https://docs.microsoft.com/en-us/azure/architecture/patterns/async-request-reply

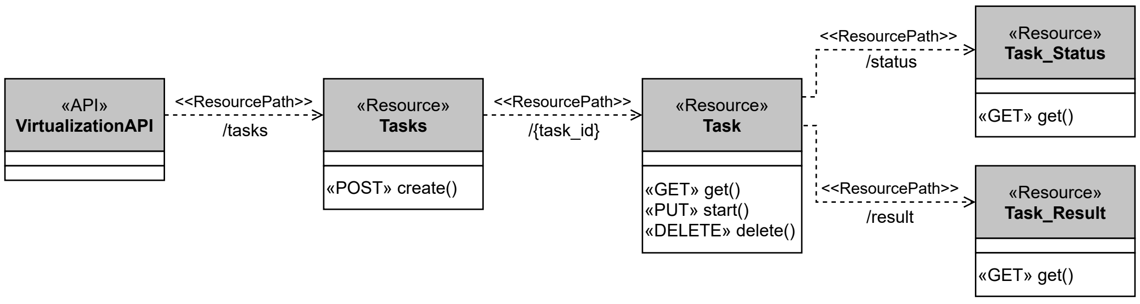


Figure 6.3: REST API of the virtualization service modeled as adjusted UML class diagram.

Listing 6.1 The minimal JSON body of a request to the Virtualization API

```
{
    "qasm": "OPENQASM 2.0; ... "
}
```

task in the database. With a second request, the client can trigger the execution of the task. The API immediately sends a response confirming the start of execution. The client can query the status of the task and retrieve the results once the task is complete. A core component of the virtualization API is the Result Fetcher. The Result Fetcher receives the results of the virtual execution environment and stores them in the database. In addition to successful results, it is also notified of errors in the virtual execution environment and marks afflicted tasks in the database as failed. The remainder of this section describes the methods of the REST API. Figure 6.3 provides an overview of the API's resources, paths, and HTTP methods.

6.3.1 Resource: Tasks

A task for the virtualized execution of a quantum circuit can be created with the *Tasks* resource. The *Tasks* resource is located under the */tasks* path. A new *Task* can be created with an HTTP POST request. It must contain a JSON object in the body that describes the quantum circuit as an OpenQASM string. Listing 6.1 shows the minimal JSON body of a valid request. The method returns an ID to identify and access the *Task*. In addition, the same method can be used to send a list of quantum circuits. Each of the list items must be a valid JSON description of a task. In this case, the method returns a list of IDs.

Furthermore, configuration data can be transmitted in addition to the pure description of the quantum circuit. The configuration changes the system's default configuration for the single request. Other requests are not affected. This way it is possible to adjust the number of shots and the configuration for the Quantum Execution Mapper and the Partitioner. For everything that is not defined in the request, the default values are used. Listing 6.2 shows a request with additional configuration. It prohibits the use of the QPU *ibmq_belem* and does not allow simulators to run this quantum circuit. It also restricts the QERs allowed to execute by a maximum qubit count and a minimum quantum

Listing 6.2 The JSON body of a request to the virtualization API with configurations

```json
{
    "qasm": "OPENQASM 2.0; ... ",
    "shots": 8192,
    "config": {
        "quantum_resource_mapper": {
            "backend_chooser": {
                "backend_black_list": [
                    "ibmq_belem"
                ],
                "allow_simulator": false,
                "number_of_qubits": {
                    "max": 5
                },
                "quantum_volume": {
                    "min": 16,
                }
            },
            "execution_types": {
                "raw": true,
                "aggregation": false,
                "partition": true
            }
        },
        "partitioner": {
            "subcircuit_max_qubits": 2,
            "max_separate_circuits": 3,
            "max_cuts": 5
        }
    }
}
```

volume. The configuration disables the aggregated execution for this request. If the system chooses the partitioned execution, it is instructed to cut the quantum circuit into at most three pieces with at most five cuts. Each subcircuit may have a maximum of two qubits.

6.3.2 Resource: Task

The *Task* resource represents a single quantum circuit created using the *Tasks* resource method presented earlier. It is located under the path */tasks/{task_id}*, where the *task_id* is a placeholder for the actual ID of the *Task*. *Task* resources provide three methods: an HTTP GET method to retrieve the *Task*, an HTTP PUT method to starts execution, and an HTTP DELETE method to remove the *Task*. The HTTP GET method returns all available information about the specified *Task*. Listing 6.3 shows a sample response of a completed *Task* with the result. The result is added once the *Task* is executed. However, the client can access the status and the result of the *Task* separately through the *Task_Status* and *Task_Result* resource. It is not necessary to query the entire *Task*.

Listing 6.3 The JSON response with the result of the virtualization API for a completed task

```
{
    "id": "60894efc0e6e9389ce977f8d",
    "qasm": "OPENQASM 2.0; ... ",
    "config": {...},
    "shots": 8192,
    "status": "done",
    "result" : {
        "0x8" : 0.1341552734375,
        "0xc" : 0.120849609375,
        "0x10" : 0.397216796875,
        "0x14" : 0.3477783203125
    }
}
```

An HTTP PUT request to the *Task* resource creates a Quantum Execution Job from the *Task* and forwards it to the virtual execution environment. Therefore, it starts the execution. The status of the *Task* is now *running*.

6.3.3 Resource: Task_Status

Task_Status resource represents the state of a *Task*. It is located under the */tasks/{task_id}/status* path for a particular *Task*. Each *Task* is in one of four states. Possible states are *created*, *running*, *done*, and *failed*. The states *done* and *failed* are final states. Once a task is in a final state, its processing is complete, and it will not receive any further updates. The HTTP GET method allows one to query the state of a *Task*. A client can poll this method periodically to check whether the *Task* is in a final state.

6.3.4 Resource: Task_Result

Task_Result resource corresponds to the result of a *Task* and is located under the path */tasks/{task_id}/result*. An HTTP GET request to this resource allows a client to get the result. The result is only available if the *Task* is in the *done* state. The result is encoded as a JSON object. Any result state that has a non-zero outcome probability is listed in the JSON object. The key is the result state encoded in hex, and the value is the probability of the state. The coding of the result is the same as in Listing 6.3.

7 Evaluation

This chapter evaluates the implementation of the virtualization concept. The focus is on evaluating the aggregation and partitioning process of the implementation. The goal is to determine how the execution of unmodified quantum circuits differs from their aggregated and partitioned execution and what causes this. For this purpose, various analysis procedures are used in the evaluation. One of these is randomized benchmarking, which is first introduced and later used to evaluate the aggregation. Then, first for aggregation and then for partitioning, the exact evaluation methods are described, and the results are presented.

The evaluation uses five-qubit NISQ devices from IBM QX. Table 7.1 shows the details of the QPUs including the quantum volume of the machines. Quantum volume is a single number used to measure the performance of a quantum computer. It considers the number of qubits and the number of operations before the qubits decohere. A higher quantum volume means that a quantum computer can solve more complex problems.

A key metric used in the evaluation is the fidelity of two quantum states. Quantum fidelity measures the proximity between two quantum states [NC10]. The fidelity value is bound between zero and one. The closer the value is to one, the more similar the quantum states are. Two equal quantum states have a fidelity value of one.

Name	**Qubits**	**Quantum volume**	**Processor type**	**Version**
ibmq_santiago	5	32	Falcon r4	1.3.19
ibmq_athens	5	32	Falcon r4	1.3.16
ibmq_belem	5	16	Falcon r4	1.0.8
ibmq_quito	5	16	Falcon r4	1.0.11
ibmq_lima	5	8	Falcon r4	1.0.8

Table 7.1: Overview of the IBMQ systems in use

7.1 Randomized Benchmarking

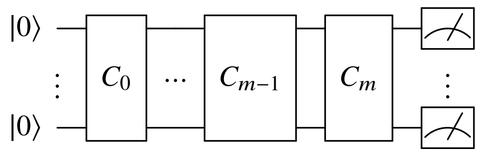
Randomized benchmarking (RB) is a technique for measuring an average error rate of a QPU [MGE11; MGE12]. RB applies sequences of random gates of different lengths, which sum to an identity operation, to an initial state. This means that after applying the entire gate sequence, the input state should theoretically be unchanged. However, due to the characteristics of today's NISQ machines, the computation is disturbed by noise, and errors accumulate. The comparison between the noisy result and the expected result, which is the input state, allows conclusions to be drawn about the QPU's average error rate.

The RB protocol consists of the following steps [MGE12]:

1. Choose a fixed initial n-qubit state $|\psi\rangle$.
2. Choose $m < M - 1$, where M is the maximum circuit depth.
3. Generate K_m gate sequences consisting of $m + 1$ operations. The sequence consists of m random operations and the inverse operation of the first m operations. The random sequence is chosen uniformly at random from the Clifford group $\mathcal{Clif}_n$ on n-qubits. This can be done by choosing each of the m operations uniformly at random from the set of generators $\mathcal{Clif}_n^{\text{gen}}$

   $$\mathcal{Clif}_n^{\text{gen}} = \{H_i, S_i, \text{CNOT}_{i,j} | i, j \in [0, n-1], i \neq j\} \tag{7.1}$$

   where i and j are the qubits' indices on which the gates act and all other qubits are unaffected by the gates. Up to a global phase, any element of the Clifford group can be generated with a sequence of the elements of $\mathcal{Clif}_n^{\text{gen}}$[HDE+06]. Thus, the first m gates are $C_0, ..., C_{m-1} \in \mathcal{Clif}_n^{\text{gen}}$ The last gate in the sequence $C_m = (C_0 ... C_{m-1})^{-1}$ is the computed reversal element that should return the qubits to the initial state. The following quantum circuit shows the resulting gate sequence.

   (7.2)

4. Execute the gate sequences on a QPU.
5. Estimate the fidelity of each gate sequence by counting how many shots return the initial state as a result. Then average over the K_m random sequences of length $m + 1$ to compute the averaged sequence fidelity $F_{\text{seq}}(m, \psi)$.
6. Repeat Steps 2 to 5 for different values of m
7. Fit the results to an exponential decay function:

   $$F(m, \psi) = A\alpha^m + B \tag{7.3}$$

   In this model, the A and B states account for the state preparation error, the measurement error, and the effect from the error on the final gate [MGE11]. The decay rate α determines the average error rate r, also called Error per Clifford (EPC) gate, in the following way [MGE11]:

   $$r = 1 - \alpha - \frac{1-\alpha}{2^n} = \frac{2^n - 1}{2^n}(1 - \alpha) \tag{7.4}$$

For the evaluation performed, the Qiskit implementation[1] was used.

[1] https://qiskit.org/documentation/apidoc/verification.html#randomized-benchmarking

7.2 Aggregation

The evaluation of the aggregation is divided into two parts. The first part analyzes the performance of aggregation with RB to measure the average error rates of QPUs. In this context, the general feasibility of the aggregation method, the influence of different QPUs, and the effects of the aggregated quantum circuits on each other are investigated. In the second part, the aggregation is evaluated in terms of the quality of the result of different quantum circuits. It analyzes the fidelity of aggregated execution of different quantum circuits.

7.2.1 Evaluation with randomized benchmarking

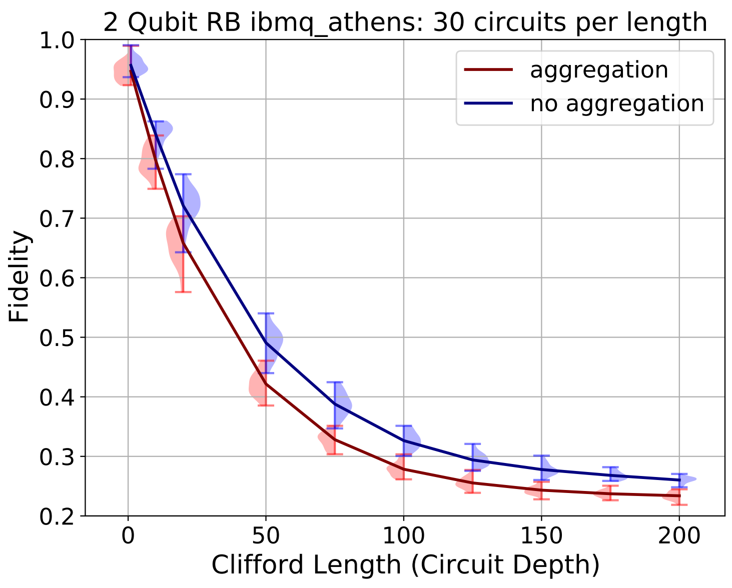
RB is a technique for measuring the average error rates of QPUs and can therefore be used to compare the average error rate on a QPU between aggregate and raw execution modes. In this section, we investigate whether aggregation of quantum circuits gives correct results on simulators. Then, we analyze how susceptible the implemented aggregation process is to the noise and crosstalk on the NISQ devices. Nevertheless, the next step first describes the exact procedure for generating the RB evaluation data.

Evaluation procedure

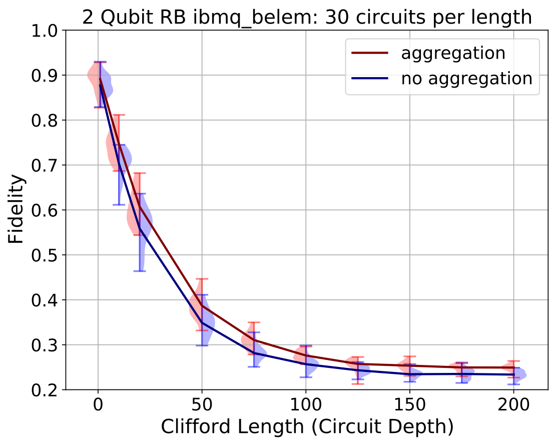
All data for the RB evaluation in this section were obtained under the following circumstances. The evaluation analyzes quantum circuits of depths 1, 10, 20, 50, 75, 100, 125, 150, 175, and 200. A Python script generated the same number of random two-qubit quantum circuits for each depth as described in Section 7.1. The evaluation is performed using IBM QX's five qubit systems from Table 7.1. In addition to the QPUs, the same evaluation was also performed once with the *ibmq_qasm_simulator*. The use of two-qubit quantum circuits allows unconstrained aggregation of the quantum circuits on each of the five-qubit QPUs. The evaluation executes each quantum circuit at least twice on each device: first without aggregation and then the same quantum circuits in aggregated pairs of two. Every execution contained 8192 shots. Unless otherwise described, quantum circuits of equal length were always combined in the aggregation procedure. The non-aggregated execution serves as a reference baseline. The evaluation assumes that the difference between the raw and the aggregated execution is the aggregation effect.

Feasibility of the aggregation procedure

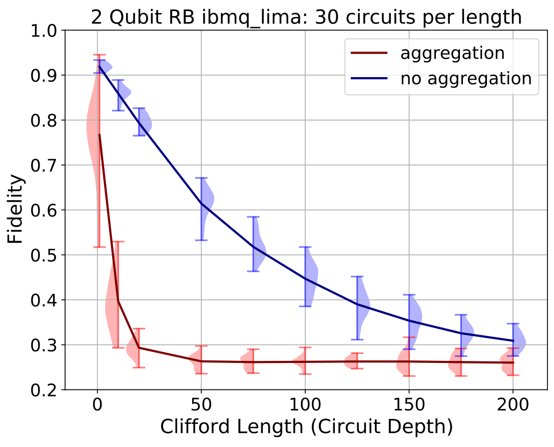
In order to test whether the implemented aggregation procedure works in the absence of quantum noise, an RB evaluation was performed using a simulator. As anticipated, no errors occur in the non-aggregated execution on the simulator. Each run in the RB successfully outputs the initial state. The aggregated evaluation of the identical circuits on the simulator leads to the same result. The results of the aggregated execution are consistent with the non-aggregated execution. Thus, the implemented aggregation method perfectly reconstructs the quantum circuit results. The implementation is feasible and works correctly in the absence of quantum noise. The remainder of the aggregation evaluation examines how the aggregation procedure performs in the presence of noise.



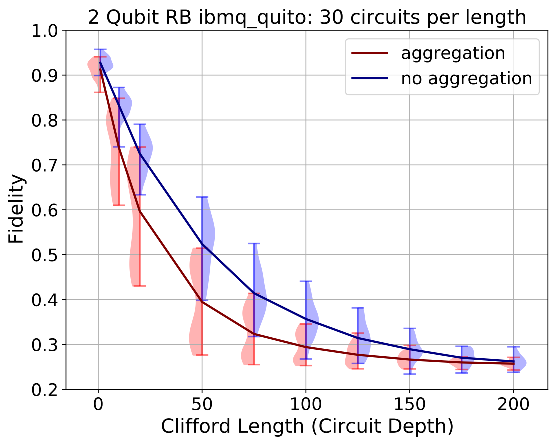
(a) *ibmq_athens*



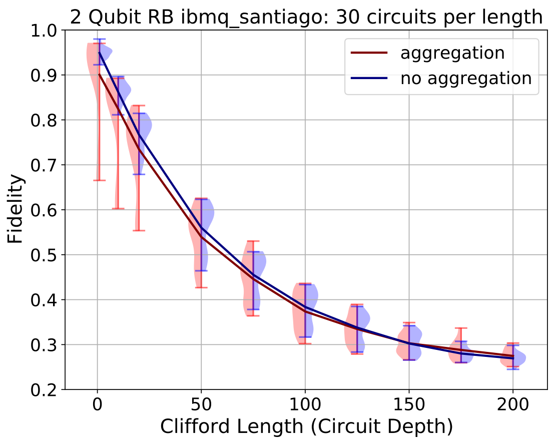
(b) *ibmq_belem*



(c) *ibmq_lima*



(d) *ibmq_quito*



(e) *ibmq_santiago*

Figure 7.1: Deviation of RB performance between QPUs from the 31.03.2021

QPU comparison

The evaluation in this section uses the RB technique described earlier to compare the aggregation on the five QPUs. Each QPU executes the same randomly generated quantum circuits, both aggregated and non-aggregated, to eliminate differences in results between QPUs due to different quantum circuits. The evaluated data consists of 30 quantum circuits per Clifford length and ten different Clifford lengths as defined above. Thus, each QPU had to perform 450 executions: 300 runs for the reference data and 150 runs for the 300 aggregated quantum circuits.

Figure 7.1 shows the RB evaluation result data for the different QPUs. Each diagram contains the following information. On the horizontal axis is the length of Clifford circuits, and on the vertical axis is the fidelity. The blue data is the reference data of the non-aggregated execution, and the red data represents the aggregated execution. Each diagram shows the data distribution of each length. The left side of the distribution shows the distribution of aggregated data, and the right side shows the distribution of non-aggregated data. The data distributions are critical to see the range of values and the spread in the range. The plotted lines connect the mean values of the data for each circuit length.

In general, all RB evaluations, whether in the aggregated or raw execution mode, show the following pattern. The fidelity starts between 0.9 and 1.0, follows an exponential decay and approaches 0.25 as the quantum circuit length increases. The average fidelity of a two-qubit circuit cannot fall below 0.25 due to noise. A fidelity of 0.25 means that each of the four possible measurement results of the two-qubit quantum circuit has the same probability. So the state of the qubit is completely decomposed and just random noise. As circuit length increases, data distributions generally become narrower.

Although all QPUs follow this general pattern, there are differences in their RB performance and especially in the RB performance of the aggregated evaluation. This is shown in Figure 7.1. As expected, the average fidelity of aggregated execution on most QPUs decreases faster than the average fidelity of raw execution. The rate at which the average fidelity of aggregation decreases relative to the reference data varies across QPUs. Contrary to expectations, for *ibmq_belem* the average fidelity of the aggregated execution decreases more slowly than the reference fidelity. In general, however, there is a correlation between the quantum volume and the aggregation method's performance. The data show that QPUs with a high quantum volume such as *ibmq_athens* or *ibmq_santiago* tend to perform better than QPUs with a low quantum volume such as *ibmq_lima*. However, the aggregation's performance cannot be predicted by the quantum volume of a QPU alone. The QPU *ibmq_belem* has a quantum volume of 16 but performs best relative to the reference data. Although, for example, *ibmq_athens* and *ibmq_santiago* both have a quantum volume of 32, the average performance on *ibmq_santiago* tends to be better.

Figure 7.2 visualizes the deviation of the aggregated execution from the raw execution of the RB. A positive value means that the aggregated execution performs worse than the raw execution. Each line represents the average difference in fidelity between the non-aggregated and the aggregated executions. The larger the value, the larger the gap between the average fidelity of the aggregated execution and the raw execution. A negative value indicates that the aggregated execution performs better than the raw execution. In most cases, the difference in performance between the aggregated and non-aggregated execution is greatest between a circuit length of 20 and 50. Shallower quantum circuits are generally less sensitive to noise, and therefore crosstalk effects are smaller. Nevertheless, the aggregated quantum circuits become more sensitive to crosstalk effects with increasing circuit

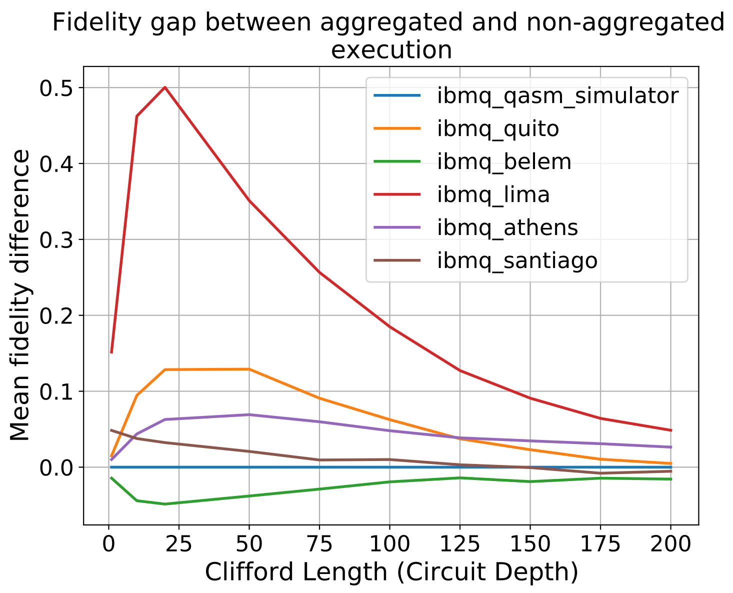


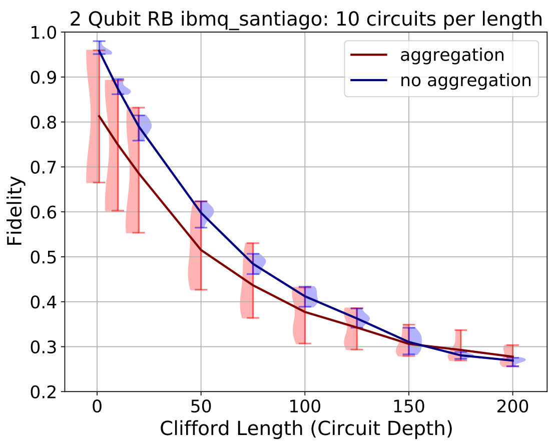
Figure 7.2: RB performance difference

Name	EPC aggregation	EPC raw	EPC difference
ibmq_santiago	0.0125	0.0118	0.0007
ibmq_athens	0.0200	0.0165	0.0035
ibmq_belem	0.0231	0.0256	-0.0025
ibmq_quito	0.0225	0.0135	0.0090
ibmq_lima	0.1022	0.0087	0.0935
ibmq_qasm_simulator	0	0	0

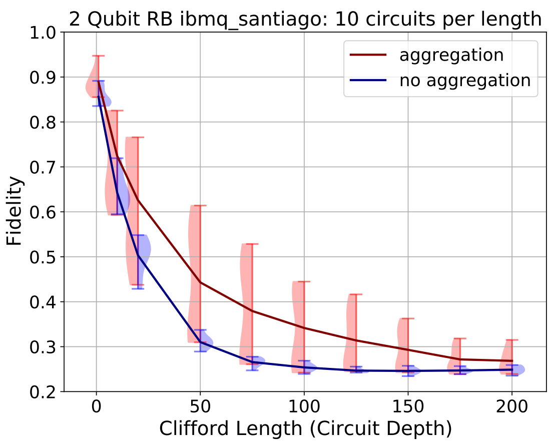
Table 7.2: RB EPC rates of the different QPUs and the simulator

depth. Quantum circuits deeper than 50 operations are so susceptible to noise that the additional effects of aggregation are of little consequence. Therefore, the performance difference becomes smaller again. Figure 7.2 also shows the performance difference of the *ibmq_qasm_simulator*. Since the simulator is not affected by crosstalk or other noise, the performance of the aggregated execution is equal to that of the circuits' raw execution. Consequently, the performance difference of the simulator is zero.

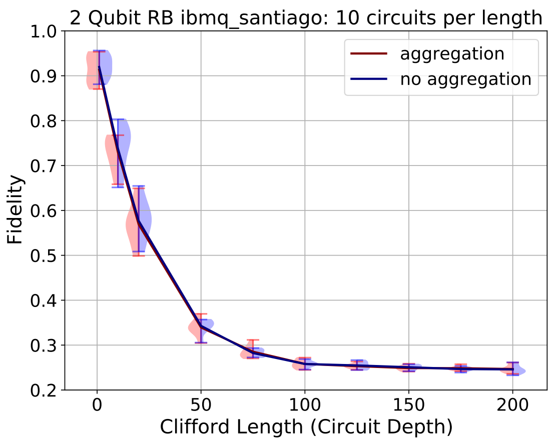
Table 7.2 summarizes the results of the comparison between the different QPUs. It shows the EPC rates for the aggregated and non-aggregated executions on the QPUs. The EPC indicates the average error per operation on the QPU. The EPC rates confirm what can be seen in Figure 7.1. The error rate of the aggregated execution mode on the QPUs is higher than the raw execution mode error rate, except for the *ibmq_belem* QPU. As Figure 7.1b shows, the EPC rate *ibmq_belem* is lower in the aggregated mode. Interestingly, a low EPC rate of raw execution does not predict a low EPC rate for the aggregated execution. For example, the QPU *ibmq_lima* has the lowest EPC rate in the raw execution but the highest EPC for the aggregated execution. The difference between the two EPC



(a) 31.03.2021 07:04



(b) 08.04.2021 09:09



(c) 08.04.2021 11:37

Figure 7.3: Variation in RB performance on one QPU at different points in time

rates indicates how much the aggregated execution differs from the non-aggregated execution. It correlates with the gap between the aggregation and raw execution curves in the graphs of Figure 7.1 as well as with the performance difference plot in Figure 7.2.

Time dependency

The data presented in the previous Section 7.2.1 were all collected at the same time. The time interval between executions on a QPU was kept as short as possible. Nevertheless, the evaluation script ran periodically during the work and performed the same evaluation several times. The resulting data show that the same QPUs perform differently at different times during the RB evaluations. Evaluations that are close in time usually lead to the same results. In contrast, results tend to diverge when there is more time between the executions. One possible reason for this could be the regular calibration of the NISQ computers. The average error rate measured with RB could depend on the calibration of the device and how long ago it was calibrated.

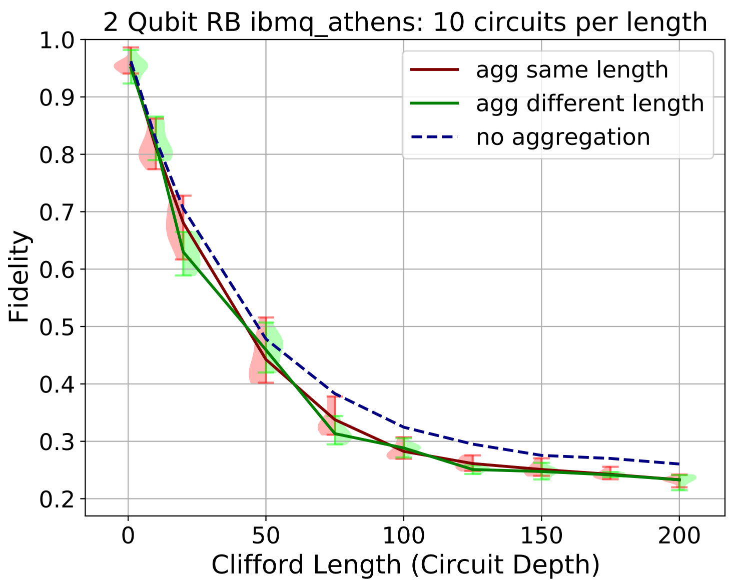
Interestingly, the differences in the evaluation due to different execution times did not always have the same effect on aggregated and non-aggregated execution. The relative performance of the two execution modes to each other was different. Again, a greater time interval between evaluations increases the likelihood of relative changes in the results. Figure 7.3 shows three executions on the *ibmq_santiago* QPU at three different time points. Their resulting graphs show three different behaviors. In Figure 7.3a, the aggregation performs worse compared to the reference data, in Figure 7.3b it performs better, and in Figure 7.3c, the performance is almost the same. It is interesting to note that in the cases where the aggregated performance deviates more widely from the reference performance, the data distribution has a higher variance. All QPUs tested exhibit the phenomenon that the quality of the aggregated results varies over time relative to the reference data. Nevertheless, the repeated evaluation confirms that the aggregated execution achieves lower average fidelity in most cases. However, it is found that the QPUs are not consistently affected by the same amount of crosstalk, which reduces the aggregated performance.

Order of aggregation

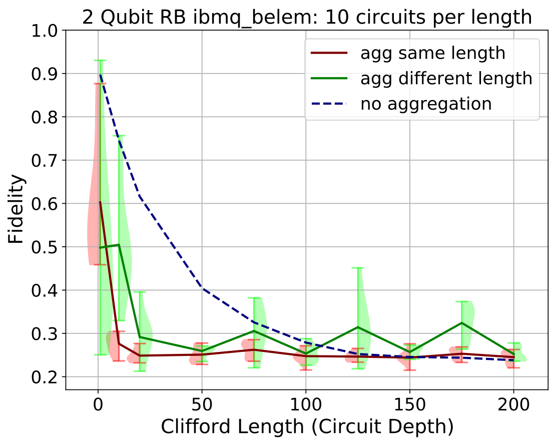
To investigate how the aggregated quantum circuits affect each other's execution result quality, the same RB quantum circuits were aggregated in different combinations. Three RB evaluations were performed using the same data. The first RB evaluation consists of unaggregated quantum circuits, which in turn serve as reference data. The other two RB evaluations consist of the same quantum circuits, but aggregated. The first aggregated RB evaluation contains aggregated quantum circuits, where circuits of the same length have been grouped together. This method has also been used in all previous evaluations. The second aggregated RB evaluation combined quantum circuits of different lengths. In this case, the quantum circuits of length 1 were aggregated with the circuits of length 10, the circuits of length 20 with the circuits of length 50, and so on. To avoid differences in the data due to temporal influences, we performed all RB evaluations in one run consecutively. For each Clifford length, we executed ten random quantum circuits. This then gives a total of 100 quantum circuits per RB evaluation and 200 executions on each QPU: 100 for each unaggregated circuit and twice 50 executions for the aggregated RB evaluations.

Figure 7.4 shows the comparison between the aggregation performance of circuits of the same length and circuits of different lengths for the different QPUs. The diagrams are similar in structure to the previous RB diagrams but contain different data. The dashed blue line shows the reference data of the raw aggregation. The red data is the same as before. It represents the aggregation of circuits of equal length. Additionally, the green-colored elements were added. They visualize the aggregation data of quantum circuits of different lengths.

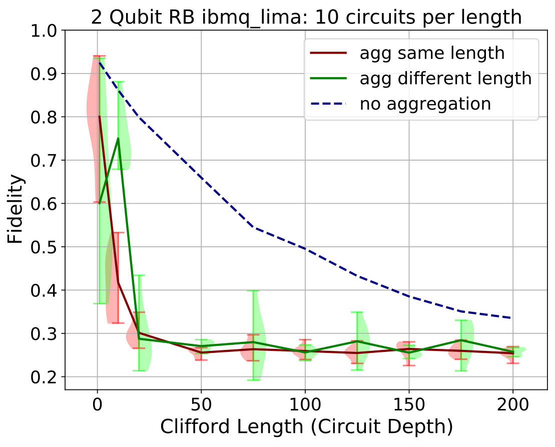
The data in Figure 7.4 shows that the aggregation of different circuit lengths roughly follows the data path of the same length aggregation. Nevertheless, while the data of the same length aggregation give a smooth curve, the data of the different length aggregation show a zigzag pattern. Generally speaking, the greater the distance between the aggregation curves and the reference curve, the more the two different aggregation curves appear to diverge from one another. An initial thought was that the reason for the zigzag pattern is that the aggregation of a shorter and a longer quantum circuit is detrimental to the shorter one because all measurements are made at the end. Then the qubits of the shorter circuit can decohere until the measurement. Das et al. [DTNQ19] used this reasoning to motivate their Delayed Instruction Scheduling for executing multiple quantum circuits of different lengths on one QPU. The pattern that the shorter quantum circuit performs worse for the same



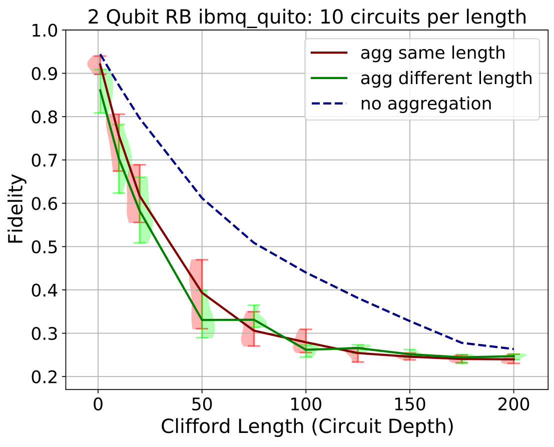
(a) *ibmq_athens*



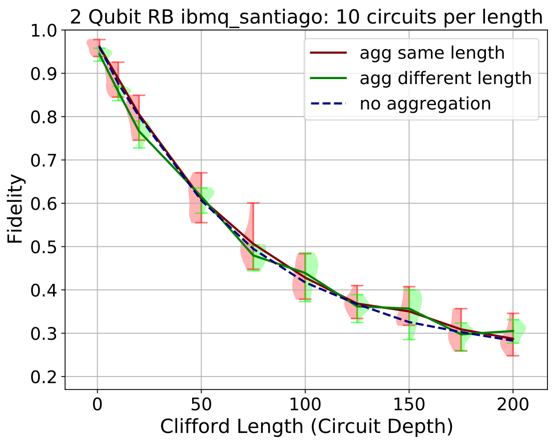
(b) *ibmq_belem*



(c) *ibmq_lima*



(d) *ibmq_quito*



(e) *ibmq_santiago*

Figure 7.4: Comparison between aggregation of circuits of the same and different lengths.

Circuit name	Depth	1-Qubit Gates	CNOTs	Result Distribution
HWEA	7	6	1	[0, 0.5, 0.5, 0]
BV	7	12	1	[0, 0, 0, 1]
QFT	9	7	2	[0.25, 0.25, 0.25, 0.25]
Supremacy Linear	9	13	1	[0.375, 0.125, 0.125, 0.375]
UCCSD	14	18	4	[0, 0, 0, 1]
Grover	20	26	2	[0, 1, 0, 0]

Table 7.3: Two-qubit quantum circuits, properties of their transpiled versions for the IBMQ QPUs, and their theoretically expected result distribution.

length can be found in the data from *ibmq_athens* (Figure 7.4a) and *ibmq_santiago* (Figure 7.4e). However, an inspection of the data from the other QPUs revealed a different pattern. The zigzag pattern on these QPUs is inverted for deeper Clifford lengths. This can bee seen in the data from *ibmq_belem* (Figure 7.4b), *ibmq_lima* (Figure 7.4c), and *ibmq_quito* (Figure 7.4d). Given this observation, it appears that on these QPUs, the shorter quantum circuit of the two aggregated circuits performs better. Nevertheless, it is critical to note that the data from QPUs *ibmq_belem* and *ibmq_lima* show a high variance in the average fidelity of the shorter aggregated quantum circuits. These findings must therefore be interpreted with caution. However, it can be observed that the aggregation of circuits of different lengths mutually affects the result. The zigzag pattern is an indication that aggregated circuits of different lengths influence each other.

7.2.2 Fidelity-based evaluation of quantum circuits

In this part, we analyze how well aggregation works with different quantum circuits. First, we present the quantum circuits for the evaluation. Then we introduce the evaluation procedure. Finally, we present the results.

Evaluated quantum circuits

We evaluated with the following two-qubit quantum circuits:

- Bernstein-Vazirani (BV) algorithm with secret 1 as input, as introduced by Bernstein and Vazirani [BV97]
- A Hardware efficient ansatz (HWEA) for the quantum approximate optimization algorithm implemented like Moll et al. [MBB+18] with circuit depth five
- A linear version of the quantum supremacy algorithm presented by Boixo et al. [BIS+18] with height 1 and depth 8
- A Quantum Fourier Transformation (QFT) circuit as described by Nielsen and Chuang [NC10]
- A circuit that performs one iteration of the Grover operator of the algorithm from Grover [Gro96]. The implemented oracle is a phase flip oracle and amplifies state 01. After one iteration, the probability for the state is already one.

- A Unitary Coupled Cluster with Singles and Doubles (UCCSD) ansatz, as described by Whitfield et al. [WBA11]

All quantum circuits except the Grover circuit were generated using the *quantum-circuit-generator*[2] with the described parameters. The Grover operator was created with Qiskit[3]. The Grover circuit contains Qiskit barriers to prevent transpiler optimizations of the oracle. Table 7.3 gives a brief overview of the quantum circuits and details of their transpiled versions.

Evaluation procedure

For each quantum circuit, we performed three different types of executions:

1. One execution on a state vector simulator
2. Unaggregated executions on a QPU
3. Aggregated executions on a QPU

The simulator's execution is needed to obtain the theoretical result state and to determine the exact probability distribution of the result. This is done once. Then the same circuit is run 100 times without aggregation on a QPU. This result is used as reference data. Finally, the quantum circuit was evaluated 100 times in aggregation mode. Therefore, the QPU must perform 50 executions. This results in 150 executions per evaluated quantum circuit and thus 900 executions for all six quantum circuits on the QPU. Each execution on the QPU contains 8192 shots. All executions of the quantum circuit were performed sequentially in one run to minimize temporal differences.

When all results are available, the data are further processed by calculating the standard classical fidelity for each result from aggregated and unaggregated executions. The classical fidelity metric is equal to the quantum state fidelity for diagonal density matrices[4]. The standard classical fidelity is obtained from the result distribution of the QPU execution p and the exact probability distribution q of the simulator for n qubits as follows:

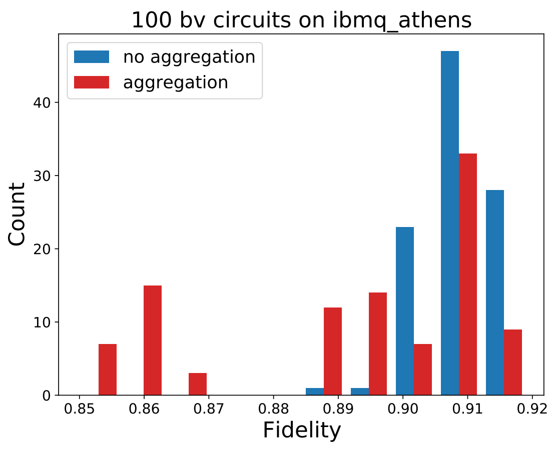
$$F(p,q) = \left(\sum_{i=0}^{2^n-1} \sqrt{p_i q_i} \right)^2 \tag{7.5}$$

Figure 7.5 shows the resulting data for *ibmq_athens* and gives an overview of the results for each quantum circuit separately. A histogram shows the fidelity values of aggregated and non-aggregated execution side by side for each quantum circuit and QPU. Results with a similar fidelity are combined and displayed as bars. The number of combined results defines the height of the bar for the given fidelity range. In the graphs, the horizontal axis represents the fidelity, and the vertical axis shows the number of executions with the given fidelity. As before, the blue-colored bars illustrate the non-aggregated execution, and the red-colored bars illustrate the aggregated execution.

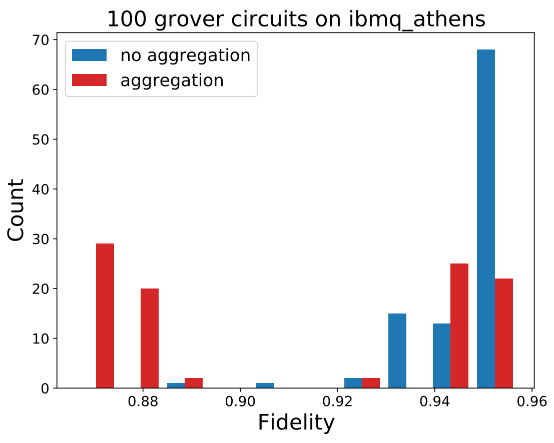
[2] https://github.com/teaguetomesh/quantum_circuit_generator

[3] https://qiskit.org/documentation/stubs/qiskit.circuit.library.GroverOperator.html

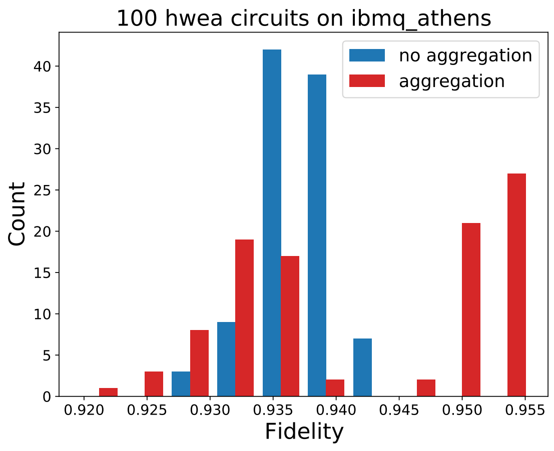
[4] https://qiskit.org/documentation/stubs/qiskit.quantum_info.hellinger_fidelity.html



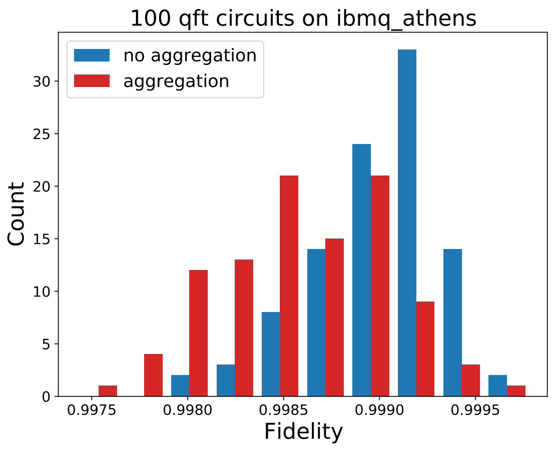
(a) BV



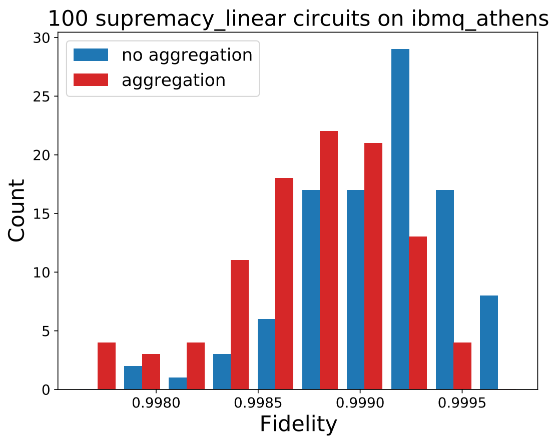
(b) Grover



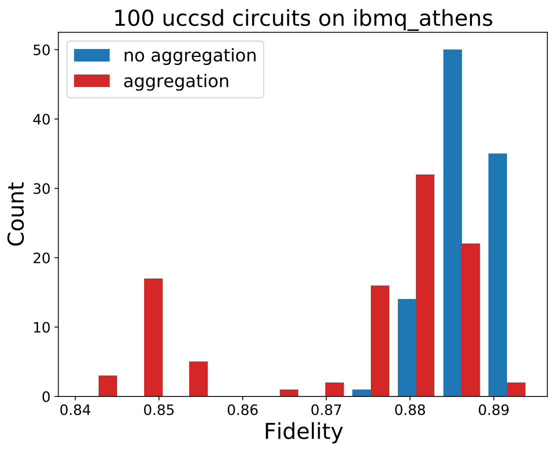
(c) HWEA



(d) QFT



(e) Supremacy Linear



(f) UCCSD

Figure 7.5: Fidelity histograms for the aggregated evaluation of different quantum circuits on *ibmq_athens*

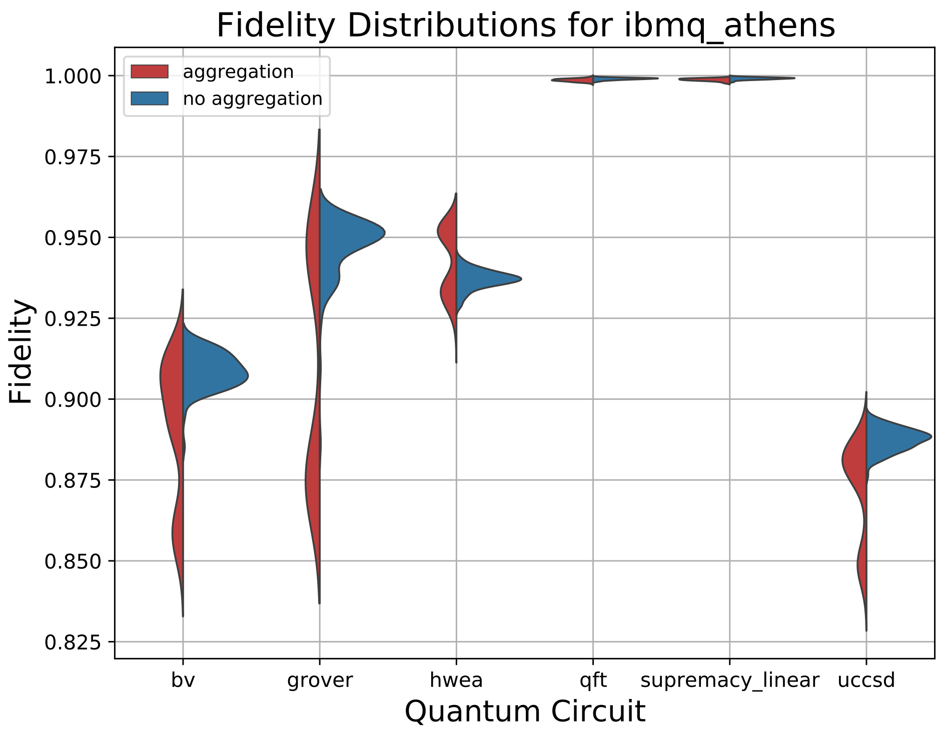
The second visualization aims at making the results of the different quantum circuits more comparable. So the QPU's chart contains the data for all quantum circuits. The plot shows the aggregated and non-aggregated fidelity distributions for each quantum circuit. A kernel density estimate approximates the underlying fidelity distribution from the actual data points. Figure 7.6 shows the visualization for the QPUs *ibmq_athens* and *ibmq_quito*. The horizontal axis lists the quantum circuits in the diagrams, and the vertical axis plots the fidelity. For each quantum circuit, the red and blue areas have the same size. However, for reasons of visibility, the areas between the different quantum circuits are scaled independently so that the maximum is always the same.

Results from *ibmq_athens*

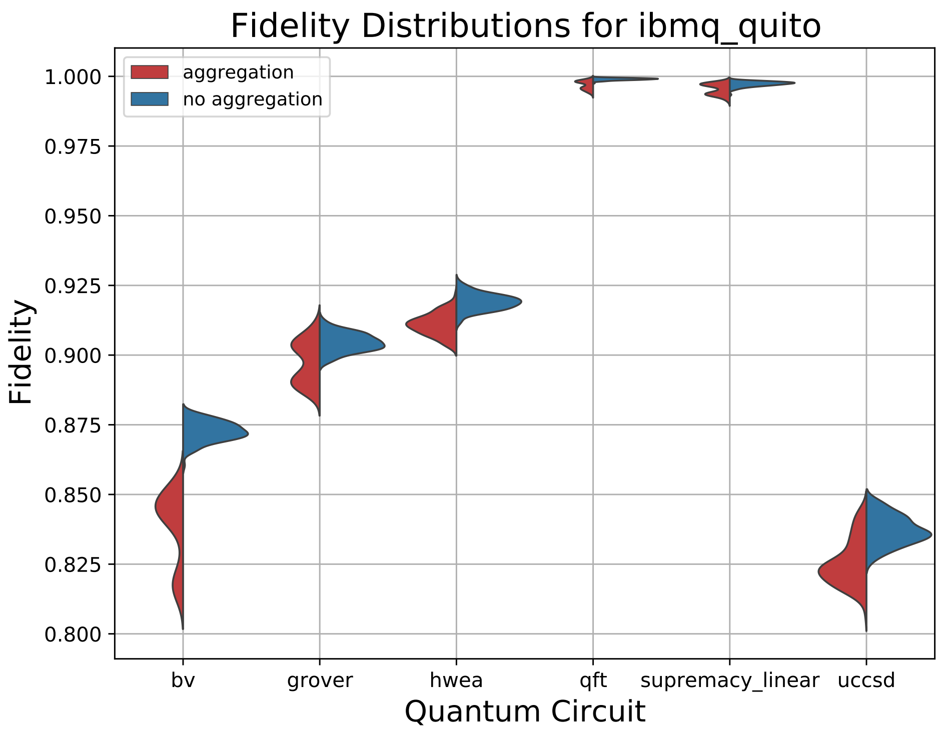
This section presents the results for the evaluation with the different circuits using the data from *ibmq_athens*. The results from each quantum circuit are depicted separately in Figure 7.5 shown earlier. Figure 7.6a relates the results of the different tests to each other. All evaluated quantum circuits achieved at least a fidelity of 0.84 in the aggregated execution and 0.87 in the non-aggregated execution. In both aggregated and non-aggregated executions, the fidelity distributions differ between quantum circuits. However, the shape of the aggregated fidelity distribution of a quantum circuit is not the same as the non-aggregated executions of the same quantum circuit. The fidelity distributions of the non-aggregated executions are more concentrated and less dispersed.

The aggregated execution of the QFT and Supremacy linear circuit performs almost as well as its non-aggregated execution. The shape of the aggregated and non-aggregated fidelity distributions are almost the same in both cases (cf. Figure 7.5d and Figure 7.5e). The fidelity distributions differ only in that the version of the aggregated circuits is somewhat flatter and slightly shifted to the left, indicating a slightly worse performance of the aggregation procedure. Nevertheless, the two quantum circuits achieved a fidelity of more than 0.995 in both execution modes. All distributions – aggregated and non-aggregated – are very dense for the two quantum circuits and have a small range of values compared to the other quantum circuits evaluated. Interestingly, the QFT and Supremacy linear circuit are the only quantum circuits evaluated where all results states have a nonzero probability (cf. Table 7.3). The expected result of the QFT circuit is the uniform distribution of result states. These circuits perform better than the others in both the aggregated and non-aggregated execution. A reasonable explanation for this phenomenon is mutually canceling error effects. If all possible states occur in the measurement result and all are equally affected by decoherence, then the collapse of one state into another can be compensated. On the other hand, if the result distribution contains only one possible state, all decoherence defects appear as errors in the result probability. Therefore, it is more challenging to achieve higher fidelity values with quantum circuits whose output contains only one measurable state. The lower achieved fidelity values of the BV, UCCSD, Grover, and HWEA circuits confirm this.

Moreover, the deviation between the aggregated and non-aggregated fidelity distributions is also more significant for these quantum circuits. The fidelity distribution of the aggregated evaluation shows two clusters, whereas the non-aggregated fidelity distribution only has one accumulation. In the BV, UCCSD, and Grover circuits, the first fidelity value accumulation of the aggregated evaluation roughly coincides with the clustering of the non-aggregated fidelity values. Their second cluster of aggregated executions has a lower fidelity. In contrast, for the two fidelity clusters of the HWEA circuit, the lower fidelity cluster matches the non-aggregated fidelity distribution.



(a) *ibmq_athens*



(b) *ibmq_quito*

Figure 7.6: Fidelity distributions for the aggregated and non-aggregated evaluation on *ibmq_athens* and *ibmq_quito*

Results from *ibmq_quito*

Figure 7.6b shows the fidelity distributions for the evaluation with equal quantum circuits for *ibmq_quito*. Their behavior on *ibmq_quito* is similar to that of *ibmq_athens*. The fidelity distributions of the non-aggregated executions are also more concentrated and less dispersed. Compared to the aggregated fidelity distributions of *ibmq_athens*, the fidelity variance of the aggregation on *ibmq_quito* is lower, but also the maximum achieved fidelity values of the different quantum circuits are lower on *ibmq_quito*.

As for *ibmq_athens*, the QFT and Supremacy linear circuits achieve the best fidelity in the aggregated and non-aggregated execution. All aggregated and non-aggregated executions of these circuits achieved a fidelity greater than 0.99. On *ibmq_quito*, the fidelity distributions of these circuits are also very dense and have a small range of values. The fidelity values of the other quantum circuits were between 0.8 and 0.93. The two-cluster pattern in the fidelity distributions is also evident in the fidelity data from *ibmq_quito*. Affected by this pattern are the quantum circuits BV, Groover, QFT, and Supremacy linear.

Results from the other QPUs

In general, the data patterns of the quantum circuits on the other evaluated QPUs are very similar. The aggregated fidelity distributions often show two clusters. One possible explanation for the two clusters is that one of the two aggregated quantum circuits performs better than the other. However, differences between the two aggregated quantum circuits cannot be the reason since we always combined the same two quantum circuits. Therefore, one explanation could be that the QPUs' qubits vary in their susceptibility to errors. Hence, it is possible that one circuit was executed on more error-prone qubits, resulting in lower fidelity compared to the other circuit executed on the less error-prone qubits.

In general, across all QPUs, there is a correlation between the fidelity of non-aggregated and aggregated execution of a quantum circuit. A quantum circuit that achieves high fidelity with low variance in the non-aggregated execution also tends to have higher fidelity with lower variance in the aggregated execution. Examples from our evaluation of this are the QFT and Supremacy linear circuit. These two circuits have in common that all result states are possible. If, on the other hand, lower fidelity with higher variance characterizes the non-aggregated execution of a quantum circuit, the same also applies to the aggregated execution. Our analysis shows that this is often the case for quantum circuits that have only one possible result state, such as BV, UCCSD, and Grover. HWEA has only two possible result states. Its performance is between the two extremes. The aggregated execution of a quantum circuit appears to amplify effects that lower fidelity and produce higher variance in the results.

7.2.3 Summary

The two evaluations show that aggregation of quantum circuits is feasible and efficient in the absence of noise. However, aggregated execution of quantum circuits generally performs worse than non-aggregated execution in the presence of quantum noise. The quality of the results for an aggregated execution of a quantum circuit depends on the QPU, the quantum circuit itself, and the other quantum circuit used for aggregation. QPUs with a higher quantum volume tend to be more suitable for aggregation. Nevertheless, the quality of the aggregated execution results for the same QPU and quantum circuit may vary over time. The data show that the performance of the non-aggregated execution predicts how well a quantum circuit performs in the aggregated execution. Quantum circuits that achieve high fidelity without aggregation also tend to maintain higher fidelity with aggregation. Furthermore, the evaluation shows that the aggregated quantum circuits influence each other's results.

7.3 Partitioning

In this section, we evaluate the implemented partitioning method of the system. It was not possible to perform an RB evaluation because the partitioning procedure could not cut all randomly generated quantum circuits. Therefore, this evaluation focuses on a fidelity-based approach as in the second part of the aggregation evaluation (cf. Section 7.2.2). This section is divided into two parts. The first part focuses on the impact of QPUs and quantum circuits on the result quality of partitioning. The second section analyzes different cuts of the same quantum circuit on the same QPU.

7.3.1 QPUs and quantum circuits

All evaluations in this section were performed using the IBM QX's five-qubit QPUs listed in Table 7.1 and the *ibmq_qasm_simulator*.

Evaluated quantum circuits

The evaluation includes the following three five-qubit quantum circuits:

- BV algorithm with secret 1111 as input, as introduced by Bernstein and Vazirani [BV97]
- A linear version of the Quantum Supremacy Algorithm presented by Boixo et al. [BIS+18] with height 1 and depth 8
- A two-bit adder circuit that adds 00 and 01. The adder is implemented like the ripple-carry addition circuit of Cuccaro et al. [CDKP04]. However, since six qubits are required for the two-bit carry ripple adder, the qubit representing the high bit for the overflow was omitted. The implemented version uses only five qubits: four qubits for the input and one for the carry.

Table 7.4 gives a brief overview of the quantum circuits, the details of their transpiled versions, and their cuts. The theoretical result of the BV and the adder circuit is a state with probability one. In contrast, the Supremacy linear circuit produces a result distribution in which all states have a non-zero probability. In this part of the evaluation, the five-qubit quantum circuits have been cut into subcircuits with a maximum of four qubits.

The other types of quantum circuits previously used for aggregation evaluation were not suitable because the partition of their five-qubit versions was unmanageable due to partitioning overhead. For example, to cut a five-qubit version of the QFT circuit into four-qubit subcircuits, at least five cuts were needed to decompose the quantum circuit into three pieces. With the various

Circuit name	Depth	1-qubit gates	CNOTs	Cuts	Parts	Subcircuits
BV	23	30	25	1	2	7
Supremacy linear	19	46	4	1	2	7
Adder	86	48	64	2	2	24

Table 7.4: Five-qubit quantum circuits, properties of their transpiled versions for the IBMQ QPUs, and details of their partitioning into four qubits.

qubit initializations and measurements required for reconstruction, the system had to execute 349 subcircuits. The partitions of the other quantum circuits have even a higher number of subcircuits. It is not feasible with the freely available IBM QX resources to evaluate a larger number of quantum circuit instances with such a high number of subcircuits per partitioned circuit instance.

Evaluation procedure

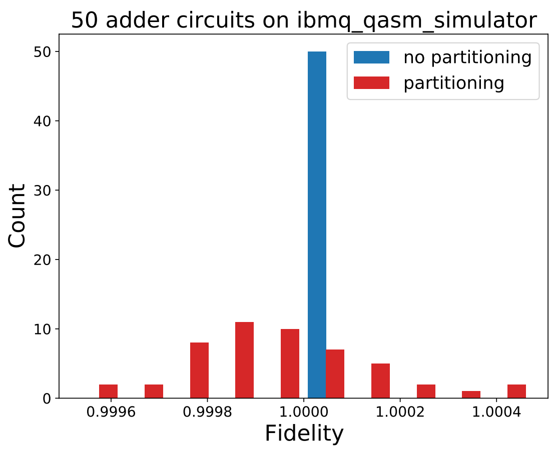
As with the aggregation evaluation, we again performed three different types of executions:

1. One execution on a state vector simulator
2. Non-partitioned executions on a QPU
3. Partitioned executions on a QPU

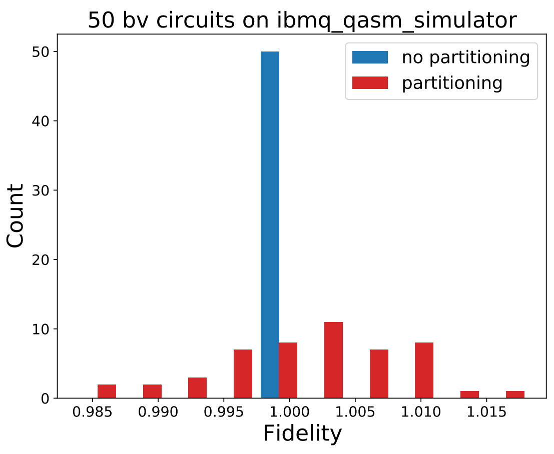
The purpose of the different types of executions is analogous to the aggregation evaluation above. However, since partitioning, unlike aggregation, creates more quantum circuits from one circuit, the number of executions on the QPU is much higher in the partitioned mode. To counteract this, we reduced the number of evaluations per quantum circuit on each QPU to 50. However, 50 evaluations of a quantum circuit in non-partitioned and partitioned modes on one QPU result in 50 non-partitioned executions plus the number of subcircuits times 50 executions for the partitioned evaluation. For the three quantum circuits, this means that we analyze 2050 executions for each QPU: 150 non-partitioned executions and 1900 executions of subcircuits. Each execution contained 8192 shots, and all executions for all quantum circuits and QPUs were performed in one run to minimize temporal differences. When all results are available, we compute the standard classical fidelity for each result from partitioned and non-partitioned evaluations as described in Equation (7.5) on page 63.

Feasibility of the partitioning procedure

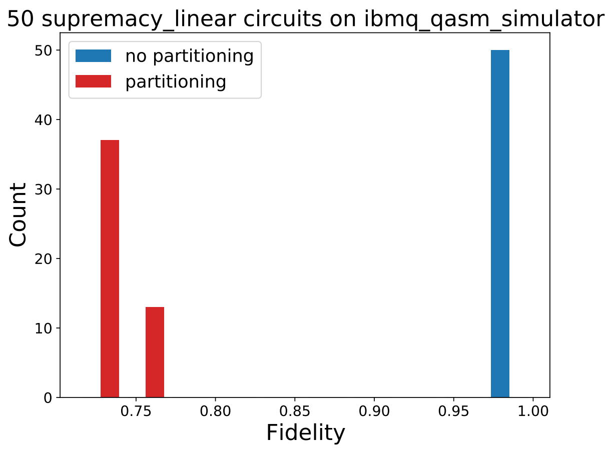
In an attempt to test whether the implemented partitioning procedure is feasible, IBM QX's cloud-based simulator *ibmq_qasm_simulator* was used in the evaluation to execute the quantum circuits without noise and errors. Figure 7.7 visualizes the results for the three quantum circuits as histograms. As expected, the simulator achieved a fidelity close to 1 for all non-partitioned executions. All evaluations of partitioned quantum circuits show a significantly higher fidelity than a random result. The average fidelity of a random result for a five-qubit quantum circuit is $2^{-5} = 0.03125$. The adder and the BV circuit both have an average fidelity of about 1, but the variance of the BV circuits is higher. Nevertheless, both circuits reproduce the correct result with a very high probability in the partitioned execution. In contrast, while the average fidelity



(a) Adder



(b) BV



(c) Supremacy Linear

Figure 7.7: Fidelity histograms for different quantum circuits on *ibmq_qasm_simulator*

of the partitioned execution of the Supremacy linear circuit is still significantly higher than the random result, it is significantly lower than the non-partitioned fidelity. The average fidelity of the partitioned Supremacy linear circuit is about 0.75. In contrast, the non-partitioned circuit achieved an average fidelity of about 0.98.

One possible reason for the significantly lower average fidelity of the Supremacy linear circuit is a limitation in the implemented classical postprocessing of the partitioning process. The postprocessing includes a method called Dynamic Definition Query that helps in reconstructing the probability distribution [TTS+21]. Postprocessing recursively applies this method to the data. Each recursion step gives a more accurate result. However, the published version of the integrated CutQC Framework [TTS+20] allows only one recursion step. This is detrimental to the results of quantum circuits such as the Supremacy linear circuit, which have more than one solution state. In general, they require more recursion steps.

Furthermore, the adder and BV data indicate that the execution on the simulator has reached fidelity values above one. However, this contradicts the definition of fidelity, which only allows values between 0 and 1. The probable cause is the imperfect probability distributions that result from postprocessing partitioning. They do not always sum to 1 and can contain probability values slightly below 0 and above 1. To apply the square root in Equation (7.5) on page 63 for classical fidelity, the evaluation rounds all negative values to 0. This distorts the result minimally so that fidelity values above one can occur.

Nevertheless, the high fidelity values confirm the feasibility of the implemented partitioning method in the absence of quantum noise and errors. However, it turns out that the average fidelity of results obtained with the partitioning method depends on the quantum circuit chosen. Our result suggests that the partitioning method works better with quantum circuits with only one expected result state instead of a result distribution where multiple outcomes are possible. Given the feasibility of the partition procedure without noise, the remainder of this section analyzes the effectiveness of the method in the presence of quantum noise on NISQ devices.

QPUs

We used the evaluation procedure described previously to evaluate the same three quantum circuits on each of the QPUs. Figure 7.8 shows the fidelity distributions of the evaluation for all QPUs for the three quantum circuits studied. Unlike previous visualizations, all red and blue areas of the distributions are the same size across all QPUs to make the data more comparable. They only vary between the quantum circuits. From the figures, it can be seen that the fidelity distributions of the partitioned executions differ between the QPUs. For example, the evaluation of the adder circuit on *ibmq_santiago* achieved a higher fidelity score than the same adder circuit on *ibmq_quito*. However, the distribution generated by *ibmq_santiago* also has a higher variance (cf. Figure 7.8). However, the fidelity distributions of the non-partitioned executions also vary between QPUs. Putting the partitioned fidelity data in relation to the non-partitioned reference data, there were no significant differences between the different QPUs that occurred across all quantum circuits. Thus, the data did not show that one QPU is better for partitioning than another.

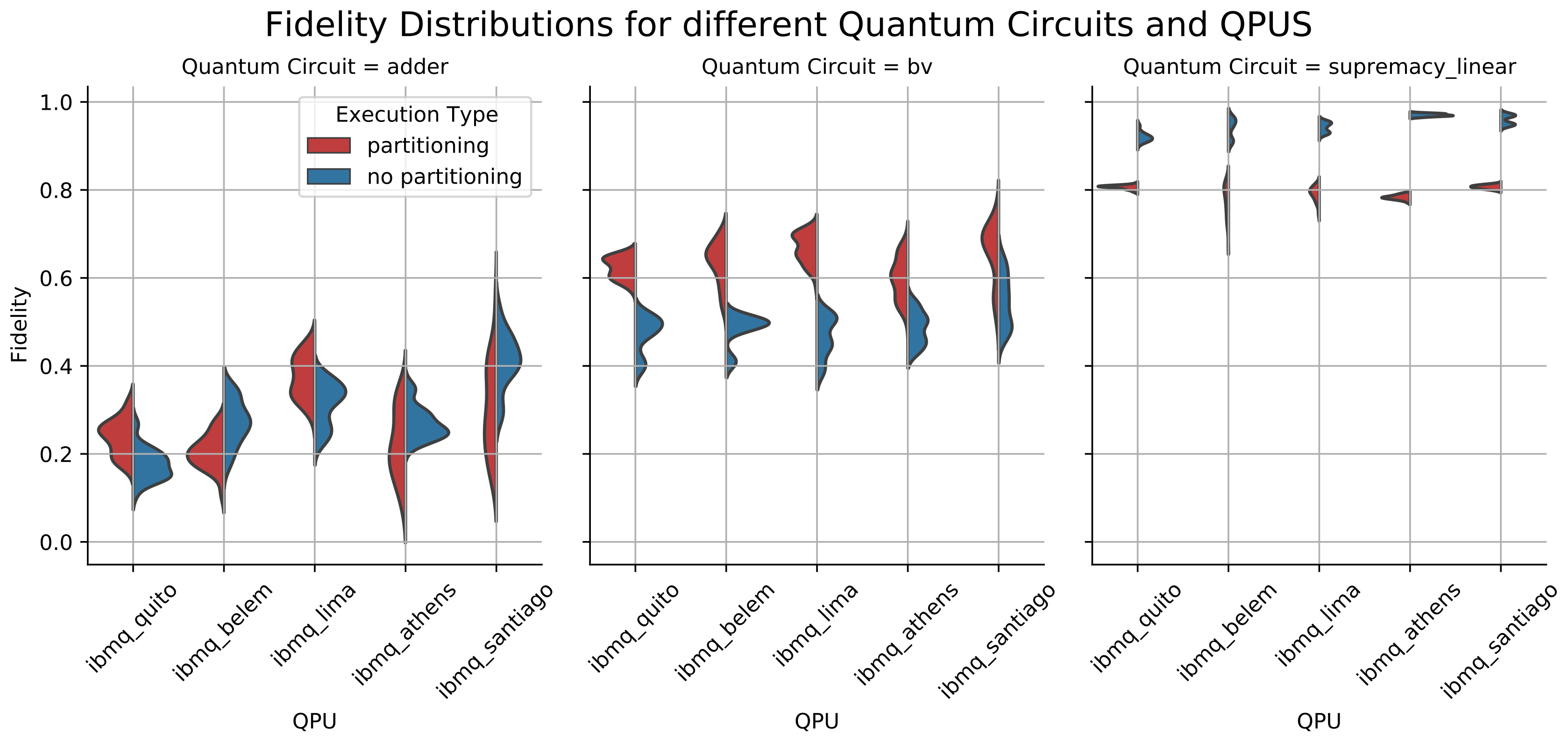
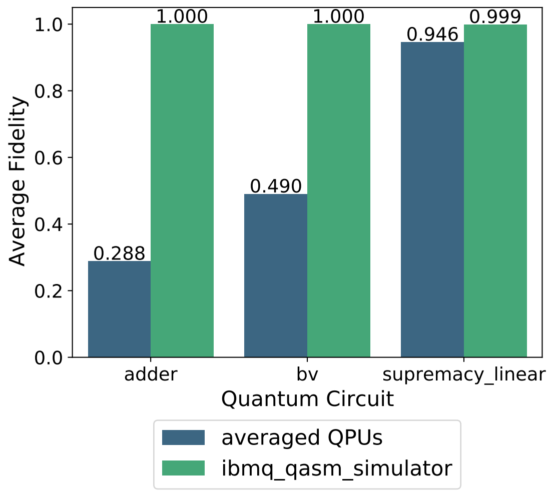
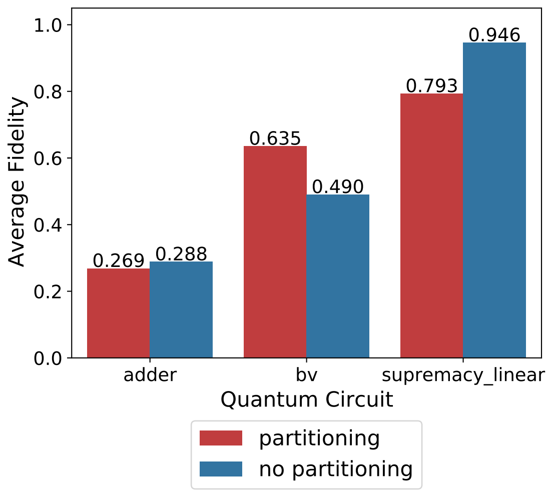


Figure 7.8: Fidelity distributions of the three analyzed quantum circuits on all five-qubit QPUs for partitioned and non-partitioned execution



(a) Comparison of the non-partitioned execution between the *ibmq_qasm_simulator* and the average over all evaluated QPUs



(b) Comparison between the partitioned and non-partitioned execution, averaged over all evaluated QPUs

Figure 7.9: Two fidelity comparisons

Quantum circuits

In this section, we consider the averaged fidelity values of the executions of a quantum circuit across all QPUs to minimize the effect from the specific QPU and focus on the influence of the quantum circuit. Compared to the evaluation on the simulator, the average fidelity of the non-partitioned circuits for the adder and BV circuit decreased significantly, as shown in Figure 7.9a. Both had an average fidelity value of 1 on the simulator. Their non-partitioned fidelity value averaged over all evaluated QPUs is 0.29 for the adder and 0.49 for the BV circuit. In contrast, the average fidelity of the non-partitioned Supremacy linear circuits does not decrease much. The average fidelity value is 0.95.

Figure 7.9b shows the averaged fidelity of the partitioned and non-partitioned execution of the three quantum circuits analyzed. The data show that the decrease in the fidelity values of the non-partitioned circuits compared to the simulator is also associated with a decrease in the fidelity of the partitioned circuits. However, the partitioned fidelity values of the different quantum circuits behave differently compared to the reference fidelity of the non-partitioned circuits.

The partitioned execution of the adder circuit achieves only a low average fidelity of 0.27 on the QPUs but performs about as well as the non-partitioned execution. Figure 7.8 shows that the adder circuit's fidelity distribution from the partitioned execution for the QPUs looks similar to the non-partitioned execution, except that the partitioned fidelity distributions of *ibmq_athens* and *ibmq_santiago* have higher variance than their non-partitioned reference distributions.

The evaluated data for the BV circuit show a different pattern. The partitioning method achieved significantly higher fidelity than the non-partitioned execution. This applies to each QPU evaluated. The average of partitioned fidelity is 0.64, and the average of non-partitioned fidelity is 0.49.

Quantum circuit	Max qubits subcircuits	Cuts	Parts	Subcircuits
BV	4	1	2	7
BV	3	1	2	7
BV	2	3	4	31

Table 7.5: Properties of different cuts of the five-qubit BV quantum circuit

Unlike the BV circuit, the data of the Supremacy linear circuit has in partitioned execution lower fidelity than in the non-partitioned execution. This applies to all QPUs evaluated. The average of partitioned fidelity is 0.79, and the average of non-partitioned fidelity is 0.95. However, it is still the quantum circuit with the highest average fidelity for both the partitioned and non-partitioned execution.

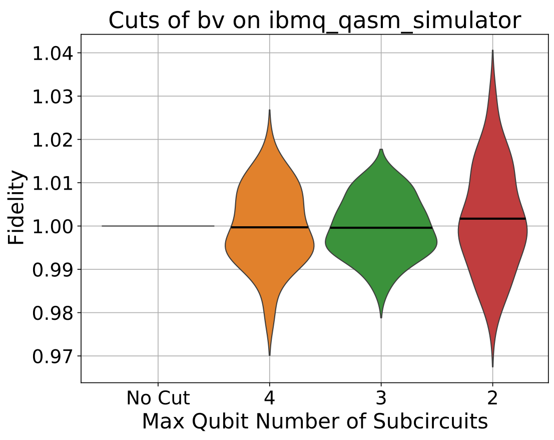
To conclude, the quality of the partitioning procedure depends on the quantum circuits. The BV circuit performs better in partitioned mode across all QPUs, while in contrast, the Supremacy linear circuit achieves higher fidelity values without partitioning.

7.3.2 Cuts

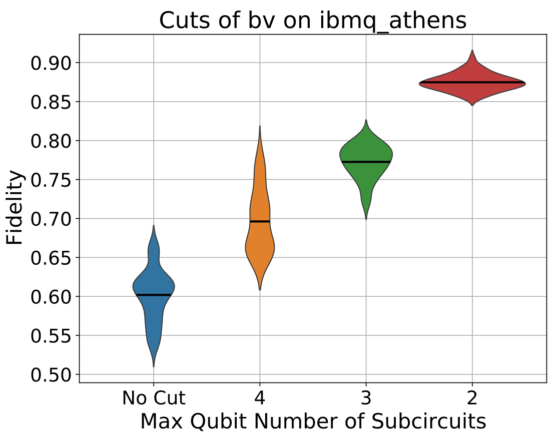
This section is the second part of the partitioning evaluation and analyzes how different quantum circuit cuts perform. The implemented partitioning procedure allows limiting the maximum number of qubits of the resulting subcircuits. The evaluation performed includes three different cuts for the five-qubit BV circuit: a cut with subcircuits of maximum width four, three, and two qubits. Table 7.5 shows the details of the cuts. It is possible to partition the BV circuit into two parts with a maximum size of four qubits with one cut. This cut generates seven subcircuits containing all measurements and qubit initializations to reconstruct the result of the original circuit. The same applies to the cut with a maximum subcircuit width of three. In contrast, a partition of the BV circuit, which has only subcircuits with two qubits, requires three cuts that separate the circuit into four parts. This results in 31 subcircuits. Each of the three cuts analyzed contained a subcircuit with the maximum qubit number.

Evaluation procedure

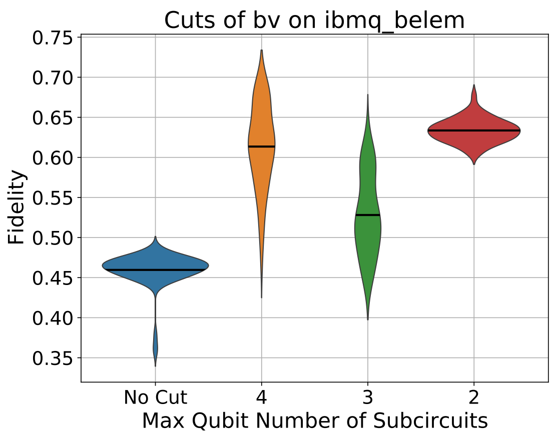
The evaluation setup is similar to the previous one. It contains one execution on a state vector simulator, 50 non-partitioned executions on a QPU, and 50 partitioned executions on a QPU for each of the cuts. This results in 2300 executions for each of the QPUs. Each execution contained 8192 shots, and all executions for all quantum circuits and QPUs were performed in one run to minimize temporal differences. When all results are available, the following evaluation step is to calculate the fidelity as described in Equation (7.5) on page 63. Figure 7.10 shows the resulting fidelity data for the *ibmq_qasm_simulator* and the five-qubit QPUs. Each visualization shows the fidelity distribution for each cut as a violin plot. The horizontal axis represents the different cutting variants, and the vertical axis the fidelity. The black line is in each violin plot marks the average fidelity.



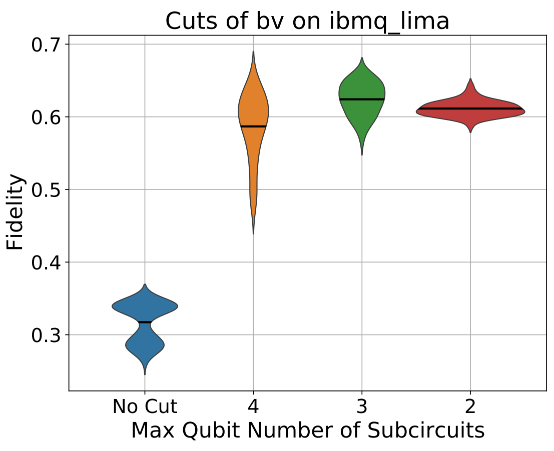
(a) *ibmq_qasm_simulator*



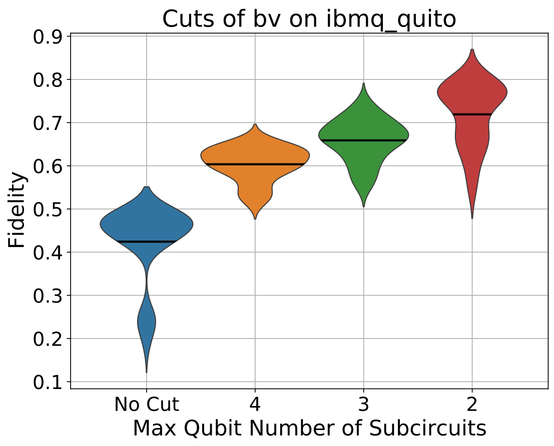
(b) *ibmq_athens*



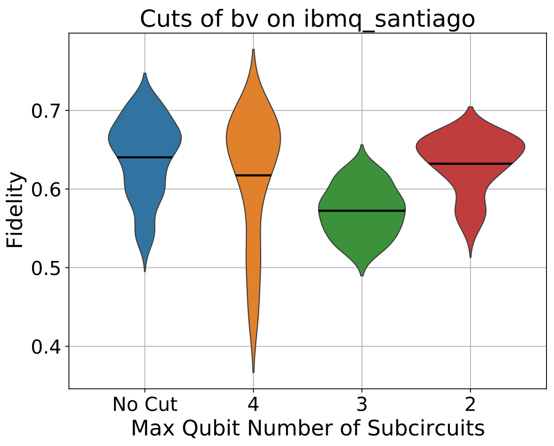
(c) *ibmq_belem*



(d) *ibmq_lima*



(e) *ibmq_quito*



(f) *ibmq_santiago*

Figure 7.10: Fidelity distributions for the different cutting variants of the BV Quantum circuit on the simulator and the QPUs

Results

Executing the three different cuts of the BV circuit on the *ibmq_qasm_simulator* revealed that in the absence of noise, all variants achieved an average fidelity of about one (cf. Figure 7.10a). Consequently, the partitioning procedure works for all cuts in the absence of quantum noise, and there is no difference in performance. However, the partition procedure produced different fidelity distributions on the noisy QPUs. The blue fidelity distribution results from the execution without partitioning and again serves as reference data. The data show that all cut variants with one exception for *ibmq_santiago* (cf. Figure 7.10f) achieved on average a higher fidelity than the reference data without partitioning.

Nevertheless, there are differences between the cuts. The resulting fidelity distributions for two-qubit cuts tend to have lower variance than the fidelity distributions for four-qubit and three-qubit cuts. The data from *ibmq_athens* and *ibmq_quito* show a different pattern: The fewer qubits the resulting subcircuits have, the higher the average fidelity. The data from the other QPUs do not show this pattern. However, for all QPUs, the average fidelity of the two-qubit cut is higher than the average fidelity of the four-qubit cut. The relative performance of the three-qubit cut varies between the QPUs.

7.3.3 Summary

The performed evaluation of the partition procedure showed that the partitioning of quantum circuits is possible. The evaluation did not identify any QPUs that are better suited for partitioned execution than others. The executed quantum circuit and the chosen cut determine much more the performance of the partitioned execution. Executions of quantum circuits that obtain higher fidelity in non-partitioned execution also tend to perform better in the partitioned mode. Nevertheless, depending on the quantum circuit, the fidelity of the partitioned execution may differ from the fidelity of the non-partitioned execution. Furthermore, the data show that, in general, cuts that produce smaller subcircuits are more favorable than cuts with larger subcircuits. However, it is not a clear-cut pattern. It seems that the quality of a cut depends on the QPU used to execute the subcircuits.

8 Discussion

In the previous chapter, the evaluation results for the quantum circuit aggregation and partitioning were presented. This chapter discusses the individual results and their relation to the overall work. In addition, the system as a whole is discussed. To this end, the feasibility of the concept is first dealt with, and only simulators are considered as the underlying execution hardware. It also looks at how the developed virtualization system behaves in conjunction with NISQ computers. Finally, this chapter discusses the general limitations of the thesis.

8.1 Concept and Implementation

The developed concept allows to abstract from the underlying quantum hardware. The drafted virtualization process decouples the logical computation of a quantum circuit from its physical execution. It uses quantum circuit aggregation and partitioning to dynamically assign the computation or parts of it to QERs. The implementation shows the feasibility of the virtual execution environment concept and proves its potential. It is shown that aggregation and partitioning of quantum circuits can be used in a system that automates their application.

The execution of the aggregated quantum circuits works flawlessly on the simulator, thus solving Challenge 1. The implemented system is able to execute two quantum circuits simultaneously as one aggregated quantum circuit and reconstruct the individual results. The aggregated results of the simulator are not inferior to the non-aggregated results. The aggregation of quantum circuits allows the system to increase the qubit utilization and throughput of the QPUs.

The evaluation of the partitioning method with simulators shows the feasibility of partitioning quantum circuits in the system. It enables computations of quantum circuits to be carried out that would otherwise be infeasible. Therefore, quantum circuit partitioning can solve Challenge 2. However, one threat to the solution is that partitioning quantum circuits introduces significant overhead. The number of subcircuits increases exponentially with the number of cuts made to partition the quantum circuit. This became clear in the evaluation carried out. Despite the use of relatively small quantum circuits, it was not possible to evaluate all the quantum circuits considered because the partitioning generated too many subcircuits for the freely available execution capacity of IBM QX. For example, partitioning the five-qubit QFT quantum circuit generates 349 subcircuits. In addition to subcircuit execution, classical postprocessing becomes more computationally intensive due to the exponential number of subcircuits.

The implemented resource mapping process employs aggregation and partitioning and automatically maps the execution of a quantum circuit to QERs. This mapping procedure frees the quantum circuit execution from the physically available quantum hardware constraints and frees the user from the necessary hardware selection. In addition, automation in the resource mapping component of the virtual execution environment allows the QER provider to better control and optimize resource usage. Therefore, the resource mapping method process solves Challenge 3 and ensures the automatic use of aggregation and partitioning for Challenge 1 and Challenge 2.

8.2 Limitations for NISQ Devices

As anticipated, errors and noise occur when using NISQ devices as the underlying execution hardware, unlike the simulators. Like normal execution, aggregated and partitioned quantum circuit execution is also susceptible to noise. However, the results of aggregated and partitioned quantum circuits differ from the unmodified execution results of a NISQ device. In general, the results from NISQ computers show that aggregated quantum circuits achieve lower fidelity than non-aggregated ones. One problem is noise, which interferes with aggregated computation more quickly and thus perturbs shallower quantum circuits than without aggregation. Furthermore, the data show that QPUs behave differently when running aggregated quantum circuits. The additional noise introduced by the aggregation varies between QPUs. Moreover, the data suggest that the aggregated quantum circuits mutually influence the quality of their output. This mutual interference shows that aggregation on NISQ devices does not allow independent and isolated execution due to crosstalk effects.

Similarly, noise from current QPUs can affect partitioning performance. Nevertheless, the data show that the result quality of a partitioned quantum circuit depends less on the choice of QPU than the aggregation of the quantum circuit. In contrast, the quantum circuit and the chosen cut seem to significantly influence the result. However, depending on the quantum circuit, the partitioned execution can achieve higher fidelity than the non-partitioned execution of a quantum circuit. Therefore, partitioning quantum circuits can improve the capabilities of NISQ devices.

Concluding, the obtained data show that the developed virtualization approach and its presented implementation are not fully applicable to modern NISQ computers. The executed quantum circuit and its mode of execution significantly influence the result. The influence of the QER and execution type used on the execution result is too substantial to equate all combinations. Especially in aggregation, the differences between the evaluated devices are too large to abstract from the execution hardware. Therefore, it is impossible to guarantee consistent quality of results and decouple the execution of quantum circuits from the underlying NISQ hardware by aggregating and partitioning quantum circuits. However, the evaluation reveals specific scenarios in which the targeted use of quantum circuit aggregation and partitioning can be helpful.

8.3 Limitations of the Implementation

The implementation represents only a prototypical system to enable the separation between logical quantum computation and its physical execution. The main focus was on the feasibility of the basic combination of different technologies to apply aggregation and partitioning of quantum circuits in an automated manner. The focus was not on optimizing individual technologies to achieve the best possible results. Therefore, their implementations lead to limitations in the system, especially in combination with NISQ devices.

A limitation of the implementation arose in the postprocessing of the quantum circuit partitioning. For some quantum circuits, it is not possible to reconstruct the exact result even on the simulator. This contradicts Tang et al. [TTS+21]. They claim to have achieved a perfect result of their partitioning method for quantum circuits with up to 100 qubits on a simulator. Lack of postprocessing capabilities in the implemented system is the probable cause of the contradiction. The integrated implementation [TTS+20] of Tang et al. [TTS+21] supports only a single Dynamic Definition recursion step in postprocessing. Several steps are meant to improve the quality of results for which a perfect result has not yet been achieved.

Moreover, the presented version of the partitioning implementation supports only successive execution of subcircuits. A single QER must execute all subcircuits resulting from a quantum circuit partition. The possibility of distributing the execution of the subcircuit to several QERs allows a more flexible and efficient computation. In addition, it enables techniques where different QERs can be deliberately used for specific parts of quantum circuits to improve the result. However, further research is needed in this area.

The implemented aggregation procedure combines the quantum circuits only rudimentarily. It does not take into account how well the quantum circuits fit together to aggregate them. Instead, it aggregates quantum circuits that are entered sequentially. Moreover, the aggregation procedure does not take into account the hardware of the executing QPU and its topology. Niu and Todri-Sanial [NT21] and Liu and Dou [LD21] present more sophisticated approaches to establish hardware-aware scheduling of aggregated quantum circuits. Implementing these methods in the virtualization system improves the result quality of quantum circuit aggregation on NISQ devices and thus the overall capability to abstract from the NISQ hardware.

Another limitation of the implementation is the initially simple resource mapping. The implemented quantum resource mapping considers only a few properties of the quantum circuit and the QER. The only considered metric of a quantum circuit is its width. Relevant QER characteristic for the implemented system are only its number of qubits and the approximated waiting time. Nevertheless, the quantum resource mapping can be extended to take into account other properties of the quantum circuits and the QERs. This would allow a more nuanced decision on the use of aggregation or partitioning in combination with the available QERs for execution. However, there are no studies on this yet.

Finally, a major limitation of the work is that virtualization is not performed directly by the cloud provider. Instead, the virtualization layer is implemented as an intermediary service that conveys quantum circuit executions and their results between the user and the cloud offering. The implementation must use the cloud interface to access the QERs. It is only possible to use the functionality of the public API over HTTP, which allows only limited access and control of the QERs. For example, there is no publicly available function to approximate the execution wait time on a QER. The implementation approximates the wait time of a QER by the job queue length, but de facto, the queue length is only a limited predictor of wait time because queue lengths are not comparable between the QERs offered by IBM QX. Moreover, exclusive access to the QERs allows for better scheduling of quantum circuit executions, as there is no need to consider executions from other customers. Furthermore, when virtualization is implemented directly by the provider, execution is not limited by a user's credits and execution limits.

9 Conclusion and Outlook

This thesis presents a virtualization approach for quantum computing to separate the logical computation of a quantum circuit from the executions on the quantum hardware. To this end, the work focuses on breaking their one-to-one relationship through the use of quantum circuit aggregation and partitioning. Quantum circuit aggregation allows one QPU to execute multiple quantum circuits simultaneously. In contrast, quantum circuit partitioning distributes the computation of one quantum circuit across multiple QPU executions. An essential component of the proposed concept is also the automation of the entire process. The resulting virtualization process automatically maps between logical computations and physical executions on hardware by systematically selecting hardware and applying quantum circuit aggregation and partitioning. Internal aggregation of quantum circuits can increase the qubit utilization and throughput of QPUs. The partitioning of quantum circuits in the process enables quantum computations that would otherwise be infeasible. Finally, the implementation of the virtualization process creates a virtual execution environment for quantum circuits. It provides a unified and hardware-independent interface for quantum computation that combines the capabilities of the underlying QERs. The virtual execution environment therefore allows focusing on the logical quantum computation and its results, while ignoring its specific physical execution.

The following evaluation of the implemented virtual execution environment proves the basic feasibility of the developed concept. It shows its potential in the absence of quantum noise. Nevertheless, the exponential computing effort associated with partitioning quantum circuits is a limitation. The processing overhead makes the partitioning method impractical for quantum circuits that require many cuts. Furthermore, the use of contemporary NISQ hardware shows that it is inadmissible to abstract from the specific execution process and at the same time guarantee a consistent result quality. The quality of the execution results differs significantly depending on the resource mapping. The QPUs analyzed have peculiarities that affect the results and make it necessary to distinguish between them. Besides the QER itself, the aggregation and partitioning of the quantum circuits also affect the results. In general, aggregate execution of quantum circuits reduces the quality of the results due to crosstalk effects on the QPU. The magnitude of the effects depends on the QPU and the aggregated quantum circuits. Quantum circuit partitioning seems to be less influenced by the choice of QPU than by the quantum circuit itself and the chosen cut. In certain scenarios, the partitioned execution provides even better results than the normal execution.

Outlook

A major limitation of the presented approach is the noise and related characteristics of current QPUs. Nevertheless, with the advancement of quantum technology, NISQ machines scale to larger sizes, are less error-prone and have fewer and fewer crosstalk effects. These improvements will enable better implementation of virtualization concepts in quantum computing in the future. Moreover, further advances in quantum circuit partitioning and aggregation will lead to better results on current NISQ devices.

As quantum computers become mainstream, cloud providers offering quantum services will have larger pools of QERs. It is then necessary to efficiently and sophisticatedly manage the increasing number of cloud-based quantum resources. An important step would be a technology that abstracts from the physical quantum hardware and separates it from the logical quantum computation. This requires virtualization concepts for quantum computing.

Bibliography

[AAA+19] H. Abraham et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2019. DOI: 10.5281/zenodo.2562110 (cit. on p. 41).

[ADK+08] D. Aharonov, W. van Dam, J. Kempe, Z. Landau, S. Lloyd, O. Regev. “Adiabatic Quantum Computation Is Equivalent to Standard Quantum Computation”. In: *SIAM Review* 50.4 (2008), pp. 755–787. DOI: 10.1137/080734479. URL: https://doi.org/10.1137/080734479 (cit. on p. 17).

[ALS+20] T. Ayral, F.-M. Le Regent, Z. Saleem, Y. Alexeev, M. Suchara. “Quantum Divide and Compute: Hardware Demonstrations and Noisy Simulations”. In: *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (July 2020). DOI: 10.1109/isvlsi49217.2020.00034 (cit. on p. 32).

[Ben80] P. Benioff. “The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines”. In: *Journal of Statistical Physics* 22.5 (1980), pp. 563–591. DOI: 10.1007/BF01011339. URL: https://doi.org/10.1007/BF01011339 (cit. on p. 15).

[BIS+18] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, H. Neven. “Characterizing quantum supremacy in near-term devices”. In: *Nature Physics* 14.6 (Apr. 2018), pp. 595–600. ISSN: 1745-2481. DOI: 10.1038/s41567-018-0124-x. URL: http://dx.doi.org/10.1038/s41567-018-0124-x (cit. on pp. 62, 68).

[BV97] E. Bernstein, U. Vazirani. “Quantum Complexity Theory”. In: *SIAM Journal on Computing* 26.5 (1997), pp. 1411–1473. DOI: 10.1137/S0097539796300921. URL: https://doi.org/10.1137/S0097539796300921 (cit. on pp. 62, 68).

[Cas17] D. Castelvecchi. “IBM’s quantum cloud computer goes commercial”. In: *Nature* 543.7644 (Mar. 2017), pp. 159–159. DOI: 10.1038/nature.2017.21585. URL: https://doi.org/10.1038/nature.2017.21585 (cit. on pp. 15, 24, 27).

[CB10] N. M. K. Chowdhury, R. Boutaba. “A survey of network virtualization”. In: *Computer Networks* 54.5 (2010), pp. 862–876. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2009.10.017. URL: https://www.sciencedirect.com/science/article/pii/S1389128609003387 (cit. on p. 25).

[CBSG17] A. W. Cross, L. S. Bishop, J. A. Smolin, J. M. Gambetta. *Open Quantum Assembly Language*. 2017. arXiv: 1707.03429 [quant-ph] (cit. on p. 41).

[CDKP04] S. A. Cuccaro, T. G. Draper, S. A. Kutin, D. Petrie Moulton. “A new quantum ripple-carry addition circuit”. In: *arXiv e-prints*, quant-ph/0410184 (Oct. 2004), quant–ph/0410184. arXiv: quant-ph/0410184 [quant-ph] (cit. on p. 68).

[CGK98] I. L. Chuang, N. Gershenfeld, M. Kubinec. “Experimental Implementation of Fast Quantum Searching”. In: *Phys. Rev. Lett.* 80 (15 Apr. 1998), pp. 3408–3411. DOI: 10.1103/PhysRevLett.80.3408. URL: https://link.aps.org/doi/10.1103/PhysRevLett.80.3408 (cit. on p. 15).

[CN05] S. Chiueh, T.-c. Nanda. *A Survey on Virtualization Technologies*. Tech. rep. 2005. URL: https://www.computing.dcu.ie/~ray/teaching/CA485/notes/survey_virtualization_technologies.pdf (cit. on p. 25).

[Dan09] J. Daniels. “Server Virtualization Architecture and Implementation”. In: *XRDS* 16.1 (Sept. 2009), pp. 8–12. ISSN: 1528-4972. DOI: 10.1145/1618588.1618592. URL: https://doi.org/10.1145/1618588.1618592 (cit. on p. 26).

[Deu85] D. Deutsch. “Quantum theory, the Church–Turing principle and the universal quantum computer”. In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400.1818 (1985), pp. 97–117. DOI: 10.1098/rspa.1985.0070 (cit. on p. 17).

[Deu89] D. E. Deutsch. “Quantum computational networks”. In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 425.1868 (1989), pp. 73–90. DOI: 10.1098/rspa.1989.0099 (cit. on p. 17).

[DiV00] D. P. DiVincenzo. “The Physical Implementation of Quantum Computation”. In: *Fortschritte der Physik* 48.9-11 (2000), pp. 771–783. DOI: https://doi.org/10.1002/1521-3978(200009)48:9/11<771::AID-PROP771>3.0.CO;2-E. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/1521-3978%28200009%2948%3A9/11%3C771%3A%3AAID-PROP771%3E3.0.CO%3B2-E (cit. on p. 23).

[DL20] X. Dou, L. Liu. “A New Qubits Mapping Mechanism for Multi-Programming Quantum Computing”. In: *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*. PACT ’20. Virtual Event, GA, USA: Association for Computing Machinery, 2020, pp. 349–350. ISBN: 9781450380751. DOI: 10.1145/3410463.3414659. URL: https://doi.org/10.1145/3410463.3414659 (cit. on pp. 28, 31).

[DTNQ19] P. Das, S. S. Tannu, P. J. Nair, M. Qureshi. “A Case for Multi-Programming Quantum Computers”. In: MICRO ’52. Columbus, OH, USA: Association for Computing Machinery, 2019, pp. 291–303. ISBN: 9781450369381. DOI: 10.1145/3352460.3358287. URL: https://doi.org/10.1145/3352460.3358287 (cit. on pp. 28, 31, 60).

[Fey82] R. P. Feynman. “Simulating physics with computers”. In: *International Journal of Theoretical Physics* 21.6 (1982), pp. 467–488. DOI: 10.1007/BF02650179 (cit. on pp. 15, 17).

[GCM17] N. M. Gonzalez, T. C. M. d. B. Carvalho, C. C. Miers. “Cloud resource management: towards efficient execution of large-scale scientific applications and workflows on complex infrastructures”. In: *Journal of Cloud Computing* 6.1 (2017), p. 13. DOI: 10.1186/s13677-017-0081-4. URL: https://doi.org/10.1186/s13677-017-0081-4 (cit. on p. 32).

[Gro96] L. K. Grover. “A Fast Quantum Mechanical Algorithm for Database Search”. In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. STOC ’96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 212–219. ISBN: 0897917855. DOI: 10.1145/237814.237866. URL: https://doi.org/10.1145/237814.237866 (cit. on pp. 15, 62).

[HDE+06] M. Hein, W. Dur, J. Eisert, R. Raussendorf, M. V. D. Nest, H.-J. Briegel. “Entanglement in Graph States and its Applications”. In: *arXiv: Quantum Physics* (2006). URL: https://arxiv.org/abs/quant-ph/0602096 (cit. on p. 54).

[JL03] R. Jozsa, N. Linden. “On the role of entanglement in quantum-computational speed-up”. In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 459.2036 (Aug. 2003), pp. 2011–2032. ISSN: 1471-2946. DOI: 10.1098/rspa.2002.1097 (cit. on p. 19).

[KMSW00] P. G. Kwiat, J. R. Mitchell, P. D. D. Schwindt, A. G. White. “Grover’s search algorithm: An optical approach”. In: *Journal of Modern Optics* 47.2-3 (2000), pp. 257–266. DOI: 10.1080/09500340008244040. URL: https://www.tandfonline.com/doi/abs/10.1080/09500340008244040 (cit. on p. 15).

[KTP+20] P. J. Karalekas, N. A. Tezak, E. C. Peterson, C. A. Ryan, M. P. da Silva, R. S. Smith. “A quantum-classical cloud platform optimized for variational hybrid algorithms”. In: *Quantum Science and Technology* 5.2 (Apr. 2020), p. 024003. ISSN: 2058-9565. DOI: 10.1088/2058-9565/ab7559. URL: http://dx.doi.org/10.1088/2058-9565/ab7559 (cit. on pp. 25, 27).

[LB20] F. Leymann, J. Barzen. “The bitter truth about gate-based quantum algorithms in the NISQ era”. In: *Quantum Science and Technology* 5.4 (Sept. 2020), p. 044007. DOI: 10.1088/2058-9565/abae7d. URL: https://doi.org/10.1088/2058-9565/abae7d (cit. on p. 24).

[LBF+20] F. Leymann, J. Barzen, M. Falkenthal, D. Vietz, B. Weder, K. Wild. *Quantum in the Cloud: Application Potentials and Research Opportunities*. 2020. arXiv: 2003.06256 [quant-ph] (cit. on pp. 24, 25, 27).

[LD21] L. Liu, X. Dou. “QuCloud: A New Qubit Mapping Mechanism for Multi-programming Quantum Computing in Cloud Environment”. In: *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 2021, pp. 167–178. DOI: 10.1109/HPCA51647.2021.00024 (cit. on pp. 31, 79).

[MBB+18] N. Moll, P. Barkoutsos, L. S. Bishop, J. M. Chow, A. Cross, D. J. Egger, S. Filipp, A. Fuhrer, J. M. Gambetta, M. Ganzhorn, A. Kandala, A. Mezzacapo, P. Müller, W. Riess, G. Salis, J. Smolin, I. Tavernelli, K. Temme. “Quantum optimization using variational algorithms on near-term quantum devices”. In: *Quantum Science and Technology* 3.3 (July 2018), p. 030503. DOI: 10.1088/2058-9565/aab822. arXiv: 1710.01022 [quant-ph] (cit. on p. 62).

[MF19] K. Mitarai, K. Fujii. *Constructing a virtual two-qubit gate from single-qubit operations*. 2019. arXiv: 1909.07534 [quant-ph] (cit. on p. 32).

[MGE11] E. Magesan, J. M. Gambetta, J. Emerson. “Scalable and Robust Randomized Benchmarking of Quantum Processes”. In: *Physical Review Letters* 106.18 (May 2011). ISSN: 1079-7114. DOI: 10.1103/physrevlett.106.180504. URL: http://dx.doi.org/10.1103/PhysRevLett.106.180504 (cit. on pp. 53, 54).

[MGE12] E. Magesan, J. M. Gambetta, J. Emerson. “Characterizing quantum gates via randomized benchmarking”. In: *Physical Review A* 85.4 (Apr. 2012). ISSN: 1094-1622. DOI: 10.1103/physreva.85.042311. URL: http://dx.doi.org/10.1103/PhysRevA.85.042311 (cit. on p. 53).

[MRN+17] M. Mohseni, P. Read, H. Neven, S. Boixo, V. Denchev, R. Babbush, A. Fowler, V. Smelyanskiy, J. Martinis. “Commercialize quantum technologies in five years”. In: *Nature* 543.7644 (Mar. 2017), pp. 171–174. DOI: 10.1038/543171a. URL: https://doi.org/10.1038/543171a (cit. on p. 15).

[NC10] M. A. Nielsen, I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. DOI: 10.1017/CBO9780511976667 (cit. on pp. 15, 17, 53, 62).

[NT21] S. Niu, A. Todri-Sanial. *Enabling multi-programming mechanism for quantum computing in the NISQ era*. 2021. arXiv: 2102.05321 [cs.AR] (cit. on pp. 31, 79).

[PG74] G. J. Popek, R. P. Goldberg. “Formal Requirements for Virtualizable Third Generation Architectures”. In: *Commun. ACM* 17.7 (July 1974), pp. 412–421. ISSN: 0001-0782. DOI: 10.1145/361011.361073. URL: https://doi.org/10.1145/361011.361073 (cit. on p. 26).

[PHOW20] T. Peng, A. W. Harrow, M. Ozols, X. Wu. “Simulating Large Quantum Circuits on a Small Quantum Computer”. In: *Physical Review Letters* 125.15 (Oct. 2020). ISSN: 1079-7114. DOI: 10.1103/physrevlett.125.150504. URL: http://dx.doi.org/10.1103/PhysRevLett.125.150504 (cit. on pp. 28, 32, 37–39).

[Pre18] J. Preskill. “Quantum Computing in the NISQ era and beyond”. In: *Quantum* 2 (Aug. 2018), p. 79. ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79. URL: https://doi.org/10.22331/q-2018-08-06-79 (cit. on pp. 15, 24).

[PTP+19] M. Perlin, T. Tomesh, B. Pearlman, W. Tang, Y. Alexeev, M. Suchara. *Parallelizing Simulations of Large Quantum Circuits*. 2019. URL: https://sc19.supercomputing.org/proceedings/tech_poster/tech_poster_pages/rpost217.html (cit. on p. 32).

[RB01] R. Raussendorf, H. J. Briegel. “A One-Way Quantum Computer”. In: *Phys. Rev. Lett.* 86 (22 May 2001), pp. 5188–5191. DOI: 10.1103/PhysRevLett.86.5188. URL: https://link.aps.org/doi/10.1103/PhysRevLett.86.5188 (cit. on p. 17).

[RP11] E. Rieffel, W. Polak. *Quantum Computing: A Gentle Introduction*. 1st. The MIT Press, 2011. ISBN: 9780262015066 (cit. on pp. 19–21).

[SAC+] M. Suchara, Y. Alexeev, F. Chong, H. Finkel, H. Hoffmann, J. Larson, J. Osborn, G. Smith. “Hybrid Quantum-Classical Computing Architectures”. In: *Proceedings of the 3rd International Workshop on Post-Moore’s Era Supercomputing* (). URL: https://par.nsf.gov/biblio/10092451 (cit. on p. 31).

[SBB+20] M. Salm, J. Barzen, U. Breitenbücher, F. Leymann, B. Weder, K. Wild. “The NISQ Analyzer: Automating the Selection of Quantum Computers for Quantum Algorithms”. In: *Proceedings of the 14th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2020)*. Ed. by S. Dustdar. Cham: Springer International Publishing, 2020, pp. 66–85. ISBN: 978-3-030-64846-6. DOI: 10.1007/978-3-030-64846-6_5 (cit. on p. 32).

[SC16] S. Singh, I. Chana. “Cloud resource provisioning: survey, status and future research directions”. In: *Knowledge and Information Systems* 49.3 (2016), pp. 1005–1069. DOI: 10.1007/s10115-016-0922-3. URL: https://doi.org/10.1007/s10115-016-0922-3 (cit. on p. 32).

[Scr17] R. Scroggins. “Emerging Virtualization Technology”. In: *Global Journal of Computer Science and Technology* (2017). ISSN: 0975-4172. URL: https://computerresearch.org/index.php/computer/article/view/1595 (cit. on pp. 15, 25).

[Sho99] P. W. Shor. “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer”. In: *SIAM Review* 41.2 (1999), pp. 303–332. DOI: 10.1137/S0036144598347011. URL: https://doi.org/10.1137/S0036144598347011 (cit. on p. 15).

[SKF+13] M. Suchara, J. Kubiatowicz, A. Faruque, F. T. Chong, C. Lai, G. Paz. “QuRE: The Quantum Resource Estimator toolbox”. In: *2013 IEEE 31st International Conference on Computer Design (ICCD)*. 2013, pp. 419–426. DOI: 10.1109/ICCD.2013.6657074 (cit. on p. 32).

[SML10] J. Sahoo, S. Mohapatra, R. Lath. “Virtualization: A Survey on Concepts, Taxonomy and Associated Security Issues”. In: *2010 Second International Conference on Computer and Network Technology*. 2010, pp. 222–226. DOI: 10.1109/ICCNT.2010.49 (cit. on p. 25).

[SPR+20] M. Sarovar, T. Proctor, K. Rudinger, K. Young, E. Nielsen, R. Blume-Kohout. “Detecting crosstalk errors in quantum information processors”. In: *Quantum* 4 (Sept. 2020), p. 321. ISSN: 2521-327X. DOI: 10.22331/q-2020-09-11-321. URL: https://doi.org/10.22331/q-2020-09-11-321 (cit. on p. 24).

[TQ19] S. S. Tannu, M. K. Qureshi. “Not All Qubits Are Created Equal: A Case for Variability-Aware Policies for NISQ-Era Quantum Computers”. In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS ’19. Providence, RI, USA: Association for Computing Machinery, 2019, pp. 987–999. ISBN: 9781450362405. DOI: 10.1145/3297858.3304007. URL: https://doi.org/10.1145/3297858.3304007 (cit. on p. 31).

[TTS+20] W. Tang, T. Tomesh, M. Suchara, J. Larson, M. Martonosi. *CutQC: Using Small Quantum Computers for Large Quantum Circuit Evaluations*. Version 1.0. Dec. 2020. DOI: 10.5281/zenodo.4329804. URL: https://doi.org/10.5281/zenodo.4329804 (cit. on pp. 41, 47, 48, 70, 79).

[TTS+21] W. Tang, T. Tomesh, M. Suchara, J. Larson, M. Martonosi. “CutQC: Using Small Quantum Computers for Large Quantum Circuit Evaluations”. In: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS 2021. Virtual, USA: Association for Computing Machinery, 2021, pp. 473–486. ISBN: 9781450383172. DOI: 10.1145/3445814.3446758. URL: https://doi.org/10.1145/3445814.3446758 (cit. on pp. 32, 37–39, 41, 47, 70, 79).

[WBA11] J. D. Whitfield, J. Biamonte, A. Aspuru-Guzik. “Simulation of electronic structure Hamiltonians using quantum computers”. In: *Molecular Physics* 109.5 (Mar. 2011), pp. 735–750. ISSN: 1362-3028. DOI: 10.1080/00268976.2011.552441. URL: http://dx.doi.org/10.1080/00268976.2011.552441 (cit. on p. 63).

[WBW15] R. Weingärtner, G. B. Bräscher, C. B. Westphall. “Cloud resource management: A survey on forecasting and profiling models”. In: *Journal of Network and Computer Applications* 47 (2015), pp. 99–106. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2014.09.018. URL: https://www.sciencedirect.com/science/article/pii/S1084804514002252 (cit. on p. 32).

[Zha18] Y. Zhang. “Virtualization and Cloud Computing”. In: *Network Function Virtualization: Concepts and Applicability in 5G Networks*. 2018, pp. 13–36. DOI: 10.1002/9781119390633.ch2 (cit. on pp. 15, 25, 26).

All links were last followed on May 19, 2021.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature