SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Quantum Circuit Transpilation: Experimental Analysis and Subarchitecture Selection

Zeynep Erdogan

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Quantum Circuit Transpilation: Experimental Analysis and Subarchitecture Selection

Quantenschaltungstranspilation: Experimentelle Analyse und Subarchitektur Auswahl

Author:	Zeynep Erdogan
Supervisor:	Professor Pramod Bhatotia, Dr.-Ing.
Advisor:	Emmanouil Giortamis, M.Eng., Francisco Romão M.Eng.
Submission Date:	15.01.2024

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 15.01.2024 Zeynep Erdogan

Acknowledgments

I would like to thank Emmanouil Giortamis, M.Eng. and Francisco Romão, M.Eng. for advising me during my thesis and helping me with all the challenges. Also, I would like to thank Professor Pramod Bhatotia, Dr.-Ing. for supervising the thesis.

I dedicate this work to my beloved husband, Ertugrul, whose love and faith in my abilities have been a constant source of motivation and strength. His presence in my life has been a source of joy and inspiration.

And to my mom, Hatice, and my dad, Fahrettin, who have given me unconditional love in every way possible. They have always prioritized my education, ensuring that I have the best opportunities to learn and grow. Their love and guidance have been the foundation of my achievements.

Abstract

Quantum computing is changing the landscape in computational capability, by surpassing classical computing in solving certain complex problems. However, the effective execution of quantum algorithms on quantum hardware presents significant challenges. Primarily in the transpilation process which is the adaptation of high-level quantum circuits to specific quantum processor architectures. This thesis explores the efficiency and scalability of quantum circuit transpilation. It mainly focuses on the optimization of quantum backend usage and the evaluation of various qubit mapping and routing algorithms.

The research investigates the performance of Qiskit's transpiler across different optimization levels. It reveals a trade-off between transpilation runtime and circuit quality. It further delves into a experimental analysis of state-of-art quantum mapping and routing algorithms. It analyzes both heuristic and constraint-based approaches. The study highlights the differences in transpilation time and the quality of the output circuits among different algorithms. Additionally, the effectiveness of employing sub-architectures in the transpilation process is assessed and It demonstrated improvements in runtime efficiency.

The findings of this research are important in understanding the complexities of quantum circuit transpilation. The study provides valuable insights into the selection of appropriate backends and optimization strategies. It contributes to the advancement of practical quantum computing applications. This thesis lays the groundwork for future advancements in quantum computing by addressing the scalability challenges and identifying efficient transpilation methodologies.

Contents

1 Introduction

Quantum computing introduces a transformative approach to computing, using quantum mechanics to tackle complex tasks that are too challenging for traditional computers. As this field progresses from theory to practice, it presents unique challenges such as executing quantum algorithms on actual quantum processors. The effective execution of these algorithms necessitates transpiling quantum circuits to fit the specific constraints of quantum hardware, a process compounded by the limited qubit connectivity, varying error rates, and architectural differences of quantum backends.

Addressing these challenges, this thesis focuses on optimizing the transpilation process for quantum circuits. It involves evaluating various mapping and routing algorithms, both heuristic and constraint-based, and exploring the efficiency of employing subarchitectures of quantum backends in transpilation. The research follows a systematic approach, beginning with an analysis of Qiskit's transpiler across different optimization levels and quantum hardware backends. This is followed by a comparative evaluation of state-of-the-art qubit mapping and routing algorithms, and an investigation into the benefits of using sub-architectures of larger quantum backends for transpilation.

The experiments entail generating random quantum circuits of varying sizes and complexities, as well as using existing benchmark quantum circuits. These circuits are then transpiled using different algorithms and backends, and the results are analyzed based on transpilation runtime, circuit quality, and scalability. The evaluation assesses the performance of various transpilation strategies, considering runtime efficiency, the number of swap gates added, and circuit fidelity. It also compares the use of full versus subarchitectures of quantum backends in the transpilation process.

This research contributes significantly to the field of quantum computing. It provides a comprehensive analysis of Qiskit's transpiler across different optimization levels and quantum backends, and a comparative evaluation of state-of-art qubit mapping and routing algorithms. The study offers insights into the scalability challenges in quantum circuit transpilation and suggests strategies to overcome them. It also enhances the understanding of quantum circuit transpilation, paving the way for more efficient and scalable quantum computing applications.

2 Background

2.1 Quantum Computing

"Quantum computing is a rapidly-emerging technology that harnesses the laws of quantum mechanics to solve problems too complex for classical computers." [5]. The unique features of quantum computing, such as superposition and entanglement, offer distinct advantages over classical computing for certain applications. While classical computers excel in tasks like data sorting, they often struggle with complex simulations of systems. Quantum algorithms introduce a approach by creating multidimensional computational spaces, enabling performing multiple test simultaneously.

One of its notable strengths lies in optimizing complex problems across diverse fields. For instance, in logistics, quantum computers can determine the most efficient routes for delivery trucks, while in finance, they excel at optimizing portfolios. However, potential of quantum computing extends beyond optimization, it poses both challenges and opportunities in the realm of cryptography. While the technology has the capability to break widely-used cryptographic algorithms, motivating the development of quantum-resistant techniques, it also introduces quantum cryptography, enhancing the security of communication channels. Moreover, the computational intensity of tasks such as simulating molecular and chemical interactions in drug discovery becomes more manageable with quantum computers, potentially accelerating breakthroughs in healthcare. In machine learning, particularly in pattern recognition and optimization tasks, quantum computing offers enhancements, and in the financial sector, it proves adept at handling intricate modeling and risk assessment calculations. Additionally, quantum computers are uniquely suited for simulating quantum systems, providing researchers with a powerful tool to delve deeper into the fundamental processes at the quantum level.

It's important to note that while quantum computing holds great potential, we're still in the early stages. Despite the speed advantages of quantum computers, tasks like Transpilation (see 2.4), which prepares circuits for execution, can often exceed the execution time the due to complexities and optimization processes. This thesis explores solutions to enhance the efficiency of quantum computing, particularly focusing on transpilation process.

2.1.1 Quantum Bits (Qubits)

In contrast to classical computers, which use bits (0s or 1s) to represent information, quantum computers use Quantum Bits (qubits). In quantum computation, the basis states $\|0\rangle$ and $\|1\rangle$ symbolize the classical bit values 0 and 1, respectively.

Qubits are kept at about a hundredth of a degree above absolute zero. At these ultra-low temperatures, materials become superconductors, allowing electrons to move without resistance and enabling key quantum computing processes [4]. Since slight environmental changes, such as heat, make qubits unstable, Qubits are prone to errors.

2.1.2 Superposition

Differing from classical bits, qubits can exist in a superposition state $|\psi\rangle = a|0\rangle + b|1\rangle$, where a and b are complex numbers subject to the constraint $a^2 + b^2 = 1$ [15]. This allows quantum computers to perform many calculations at once, potentially offering a significant speedup for certain types of problems.

2.1.3 Entanglement

Another important feature is called Entanglement where qubits become correlated and the state of one qubit is dependent on the state of another, regardless of the distance between them. Just as a single qubit can be in a superposition of 0 and 1, a register of n qubits can be in a superposition of all 2^n possible values. Any two-qubit state can be represented as $a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle$, where a, b, c, and d are complex numbers such that $|a|^2 + |b|^2 + |c|^2 + |d|^2 = 1$. [15]

2.1.4 Quantum Operations

There are two types of operations a quantum system can undergo: measurement and quantum state transformations. In classical computing, universal sets of gates exist, allowing the execution of any computation through a series of these gates. Similarly, quantum computing involves universal sets of primitive state transformations known as quantum gates. With a sufficient number of quantum bits, it becomes feasible to build a universal quantum Turing machine. [15]. For example, The Controlled NOT (CNOT) gate, functions on two qubits in the following manner: it modifies the second bit if the first bit is 1, and otherwise, it leaves the second bit unaltered [15].

Another quantum operation is called measurement. Most quantum algorithms involve a sequence of quantum state transformations followed by a measurement. [15]. When measuring a single qubit in a state $|\psi\rangle = a|0\rangle + b|1\rangle$, the probability of the outcome 0 is $|a|^2$, and the probability of outcome 1 is $|b|^2$.

On IBM quantum machines, two-qubit operations are carried out through coupling links connecting specific pairs of qubits. Due to practical limitations in superconducting quantum computers, not all qubits can be interconnected freely, leading to the adoption of restricted network structures like Mesh, allowing connectivity only between neighboring qubits. These network constraints influence which qubits can be entangled. Fortunately, SWAP operations exist, enabling the movement of qubits between locations and facilitating the entanglement of any two arbitrary qubits. Even if a quantum machine lacks a native SWAP instruction, achieving it is possible through the utilization of three CNOT gates. [18]

2.1.5 Quantum Circuits

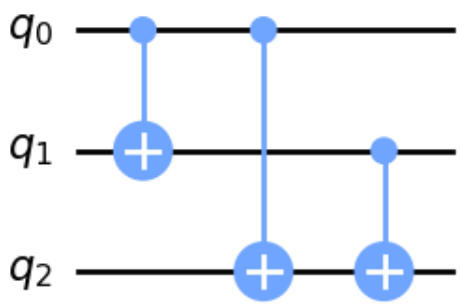
Quantum circuits form the core of quantum computing, utilizing qubits and a series of quantum gates to facilitate complex computational tasks. Unlike classical bits, qubits can exist in multiple states simultaneously due to superposition, allowing for more intricate computations. The structure of a quantum circuit involves strategically placing quantum gates to manipulate these qubits. These gates, like the Pauli-X, Y, Z gates, the Hadamard gate, and the CNOT gate, each have unique functions, from changing qubit states to creating entanglements between qubits. Designing a quantum circuit thus requires a careful selection of these gates to achieve the desired computational outcomes. This design process is intricate, considering not only the quantum mechanical principles but also the specific capabilities and limitations of the quantum hardware being used.

```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[3];
creg c[3];

cx q[0],q[1];
cx q[0],q[2];
cx q[1],q[2];
```

(a) QASM File



(b) Quantum Circuit

Figure 2.1: Example Quantum Circuit with 3 CNOT Gates

In quantum computing, a Quantum Assembly Language (QASM) file is a script used to describe quantum circuits in a standardized, low-level language. It specifies

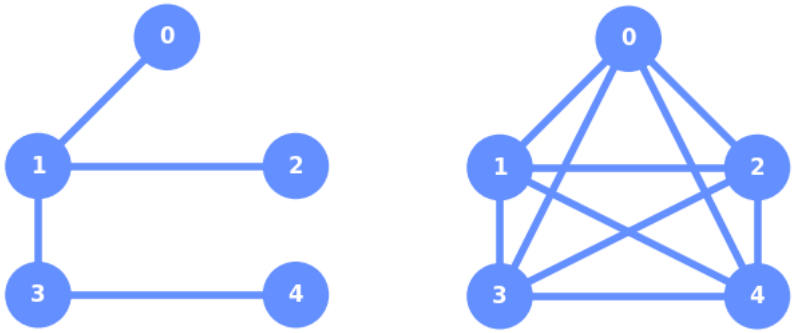
quantum gates, their arrangement, and application to qubits (quantum bits), making it an essential tool for programming quantum computers and simulators. With its human-readable format, QASM bridges the gap between abstract quantum algorithms and practical implementation on quantum hardware. Figure 2.1 Displays an example quantum circuit with 3 CNOT gates and It's corresponding QASM file in the left.

2.2 Noisy Intermediate-Scale Quantum (NISQ) Computers

Achieving reliable quantum information processing requires keeping the system almost perfectly isolated while enabling strong qubit interactions and external control. Overcoming these conflicting demands has demanded years of progress in materials, control, and fabrication. The vision for scalability involves quantum error correction, which encodes information in a highly entangled state, protecting it from environmental interference. However, the substantial overhead cost of implementing Quantum Error Correction (QEC), requiring many additional physical qubits (QEC requires a large number of physical qubits (10x-100x)to encode one fault-tolerant bit [18].), suggests that practical, error-corrected quantum computers are not expected in the near term [14]. For example implementing extensive quantum applications, like Shor's factoring algorithm, necessitates a quantum computer equipped with millions of qubits [18]. So John Preskill comes with the term Noisy Intermediate-Scale Quantum (NISQ) [14]. Here "intermediate scale" refers to a number of qubits ranging from dozens to hundreds.

A notable characteristic of NISQ (Noisy Intermediate-Scale Quantum) computers is their lack of full connectivity among qubits, as depicted in Figure 2.2 which contrasts the IBM Vigo Backend with a simple 5-node connected graph. Unlike the idealized model of a fully connected graph, where every node (qubit) can directly interact with every other, real-world NISQ computers often have limited connectivity. This means that not all qubits are directly linked, leading to challenges in implementing quantum circuits that require interactions between non-adjacent qubits.

This limited connectivity necessitates the transpilation process in quantum computing. Transpilation involves rewriting quantum circuits to fit the specific architecture of a quantum processor. It often requires inserting additional operations, like SWAP gates, to facilitate interactions between qubits that are not directly connected. The transpilation process aims to adapt a quantum algorithm to the constraints of a specific quantum device, optimizing it for the available qubit connections and minimizing the impact of this limited connectivity on the algorithm's performance and accuracy. This step is crucial in the current NISQ era, where achieving effective quantum computation depends heavily on how well algorithms are mapped and adapted to the specific qubit architectures of available quantum hardware.



(a) IBM Vigo Backend Architecture (b) Connected Graph with 5 Nodes

Figure 2.2: IBM Vigo Backend vs Connected Graph with 5 Nodes

As of December 2023, IBM has achieved a significant milestone in quantum computing by releasing their latest processor, Condor, featuring 1121 qubits. Alongside this, IBM has introduced Heron, a 133-qubit processor that demonstrates a 3-5x improvement in device performance over their previous flagship. These advancements follow the releases of the 433-qubit Osprey and the 127-qubit Eagle processors within the past two years. Additionally, IBM offers smaller processors with 7, 27, and 33 qubits.

In this thesis, the experiments are conducted exclusively on IBM machines; however, it is important to note that there are several other key players in the quantum computing arena. Companies like Google, Rigetti Computing, and IonQ, as well as academic and government research institutions, are also making significant contributions to the field. These providers are developing their own quantum processors and technologies, each with unique approaches to quantum computing. For instance, Google's quantum computing efforts are focused on superconducting qubits, similar to IBM, while IonQ is advancing quantum computing with trapped ion technology. Rigetti Computing is also known for its work with superconducting qubits and has been actively involved in developing quantum cloud services.

2.3 Fidelity of NISQ Computers

NISQ computers, while representing a significant advancement in quantum computing, inherently suffer from various types of errors due to their noisy nature. These errors can occur during different operations and processes within the quantum computer, including single qubit operations, two-qubit operations like CNOT gates, and during qubit retention or readout phases. Understanding and mitigating these errors is crucial

for improving the fidelity and overall performance of NISQ systems.

- **Single Gate Errors**: These errors occur during the operation of single-qubit gates, impacting the accuracy of individual qubit manipulations. Examples of single-qubit gates where such errors might occur include the Pauli gates (X, Y, Z), which flip the state of a qubit, the Hadamard gate (H), which creates a superposition of states, and the Phase Shift gates (like S and T), which change the phase of a qubit's state. Errors in these gates can lead to incorrect quantum states, thus affecting the overall computation process and its outcomes. For instance, an error in a Hadamard gate could result in a qubit being in the wrong superposition state, while an error in a Pauli gate could incorrectly flip a qubit's state, both leading to significant inaccuracies in the execution of quantum algorithms.
- **Two-Qubit Gate Errors**: This error rate is specific to operations involving two qubits and is a crucial metric in quantum computing. Errors in two-qubit gates, such as the Controlled-NOT (CNOT) gate, are particularly significant as they directly affect the entanglement process and the overall integrity of quantum computations. For instance, in the CNOT gate, any error can lead to incorrect entanglement between the qubits, disrupting the intended quantum operation and potentially leading to erroneous computational results.
- **Retention Errors (or Coherence Errors)**: A qubit's ability to retain information is time-bound, known as the Coherence Time. Retention errors emerge in two forms:
  - **T1 Coherence Time**: This metric denotes the time constant associated with a qubit's natural decay from a high-energy state ($|1\rangle$) to a low-energy state ($|0\rangle$), indicating the time for a qubit's natural relaxation.
  - **T2 Coherence Time**: This time constant is linked to phase errors resulting from a qubit's interaction with its environment, reflecting the duration a qubit remains unaffected by environmental factors.[18]
- **Readout Errors**: Occurring during the measurement phase, readout errors affect the accuracy of interpreting the final state of qubits after computation.

Figure 2.3 presents the average calibration data of the IBM Kolkata machine for October 2023, highlighting three key metrics: **readout error**, **Hadamard (H) gate error rate**, and **CNOT gate error rate**. These metrics provide insights into the error rates for fundamental operations within the quantum processor.

The calibration data shown in Figure 2.3 also reveals a notable aspect: certain qubit links, for example between 4-7 and 21-23, exhibit a 100% error rate in their CNOT gate

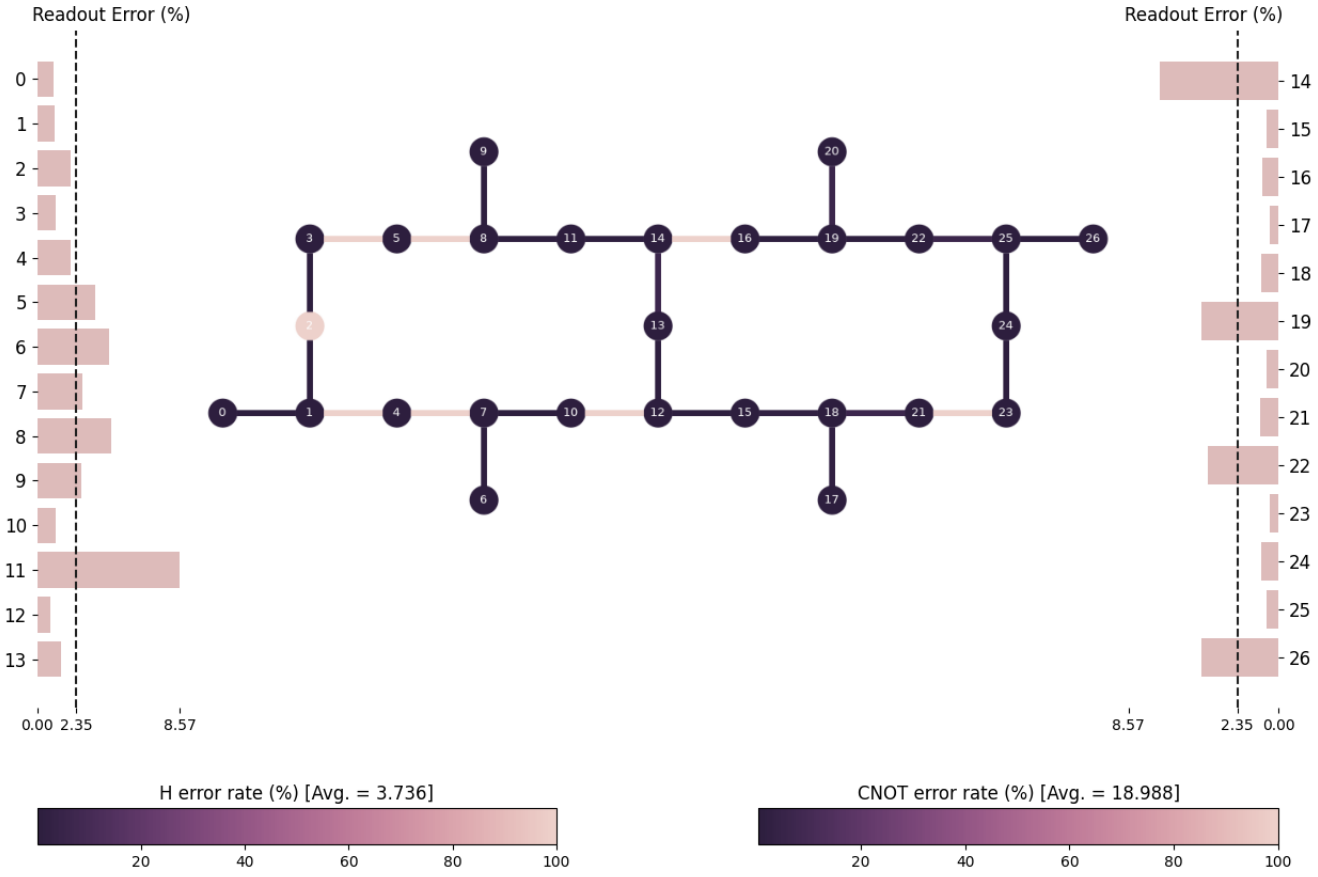


Figure 2.3: Average Error Rates of IBM Kolkata during October 2023

operations. This suggests that every attempt to execute a Controlled-NOT operation between these qubits is unsuccessful. Such a high error rate indicates possible hardware malfunctions or defects in the quantum links between these qubits. It could also suggest critical misalignment or calibration errors within these parts of the quantum system. This level of error severely limits the processor's functionality for any quantum algorithm requiring reliable CNOT operations between these qubits, underlining the challenges in maintaining high fidelity in quantum computations.

2.4 Transpilation

Transpilation in quantum computing is the complex process of adapting and refining a quantum circuit to both align with the specific architecture of a chosen quantum device and to enhance its execution efficiency on current, error-prone quantum systems.

The procedure of transpilation involves several transformations, to ensure the quantum circuit's compatibility with the target device and to minimize the impact of noise on the circuit's output. The adaptation and optimization of quantum circuits for specific hardware entail intricate logic flows that are not necessarily linear; they can include iterative sub-processes, conditional branches, and various sophisticated steps.

The stages of transpilation include: Initialization Stage, Mapping Stage, Routing Stage, Translation Stage, Optimization Stage, Scheduling Stage. This thesis focuses on Mapping and Routing Stage which solves the qubit mapping and routing problem.

The qubit mapping and routing problem in quantum computing presents significant challenges. It involves tasks akin to sub-graph isomorphism, where logical qubits are assigned to physical ones, and token-swapping, which relates to determining the optimal sequence of swaps. These tasks are known to be NP-hard or potentially PSPACE-complete, adding to the complexity of the qubit mapping and routing problem. [2]

2.4.1 Mapping Stage

The Mapping Stage, also known as Layout Stage, is a critical phase in the transpilation process where virtual qubits from the quantum circuit are mapped onto the physical qubits of the quantum processor. This stage is crucial because the physical architecture of quantum devices often imposes constraints on which qubits can interact directly.

In this stage, the transpiler takes into account the topology of the quantum device, which includes understanding the connectivity between qubits and the specific characteristics of each qubit. The aim is to find an optimal mapping that minimizes the need for additional operations, such as SWAP gates, which are required when the desired qubit interactions are not directly supported by the hardware's native connectivity.

The effectiveness of the layout greatly influences the performance and error rates of the quantum computation. A well-optimized layout can significantly reduce the circuit depth and the number of additional operations, thereby enhancing the overall fidelity of the quantum computation.

2.4.2 Routing Stage

Following the Mapping Stage, the Routing Stage addresses the challenge of adapting the quantum circuit to the specific connectivity constraints of the quantum processor. Even with an optimized layout, it is often necessary to insert additional gates into the circuit to enable interactions between qubits that are not directly connected on the device.

In this stage, the transpiler injects SWAP gates into the circuit, rearranging the positions of qubits so that each CNOT gate in the circuit acts on qubits that are physically connected on the device. The routing process requires careful consideration to balance between minimizing the number of additional gates and maintaining the overall integrity of the quantum computation.

The Routing Stage is important for executing complex quantum algorithms on actual quantum hardware, where direct qubit-qubit interactions are limited by the physical layout of the device. An efficient routing algorithm can substantially reduce the computation time and error rates, a key factor in achieving practical and reliable quantum computation on current NISQ devices.

Figure 2.4: Transpiled Quantum Circuit with an Added Swap Gate

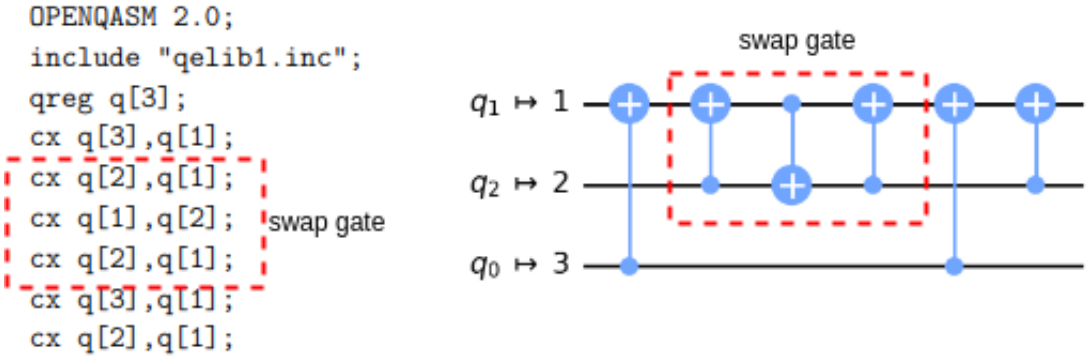


Figure 2.4 illustrates the transpiled version of quantum circuit shown in Figure 2.1. The transpilation process adapts the original circuit to be compatible with the specific architecture of the IBM Vigo backend.

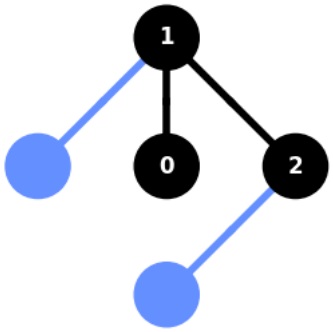


Figure 2.5: Placement of Transpiled Circuit in IBM Vigo Backend

The layout of the transpiled circuit for the Vigo backend is presented in Figure 2.5. In this layout, we observe a strategic mapping of the circuit's logical qubits to the physical qubits of the Vigo backend: logical qubit 0 (q0) to physical qubit 1, logical qubit 1 (q1) to physical qubit 2, and logical qubit 2 (q2) to physical qubit 3. Notably, this mapping strategy leaves physical qubits 0 and 4 unused.

Transpilation process adds a swap gate which is shown as 3 CNOT gates between the q1 and q2. This modification is necessitated by the absence of a direct connection between physical qubits 1 and 3 in the Vigo backend's architecture. If there had been a direct link between these physical qubits, the SWAP gate would have been redundant, potentially simplifying the circuit and reducing possible sources of error. This example demonstrates the important role of transpilation in quantum computing, showing how it effectively adapts a quantum circuit to align with the connectivity constraints and architectural specifics of a quantum processor.

2.5 Qiskit: A Framework for Quantum Computing

Qiskit stands out in the quantum computing landscape as a vital open-source software development framework. Developed by IBM, it empowers a broad spectrum of users, from beginners to advanced researchers, enabling them to build, simulate, and run quantum circuits on actual quantum hardware.

3 Overview

The system is designed to address the challenges of qubit mapping and routing in quantum computing. It encompasses both software algorithms and hardware considerations to optimize quantum circuit transpilation and execution. The system integrates state-of-the-art algorithms for qubit mapping and routing, assessing their performance across different quantum hardware architectures. This chapter provides an overview of the system, outlining its design goals, workflow, and the various components that constitute the system.

3.1 Design Goals

The primary design goals of the system include:

Efficiency: Maximizing computational efficiency in the transpilation process and the execution of quantum circuits.

Scalability: Ensuring the system can handle quantum circuits of varying sizes and complexities.

Adaptability: Providing effective solutions across different quantum hardware backends, each with unique characteristics and constraints.

Optimality: To achieve the best possible quantum circuit quality, in terms of both fidelity and resource utilization.

3.2 System Workflow

The workflow of the system is designed to utilize the capabilities of quantum computing efficiently. It has several stages, each dedicated to a specific aspect of quantum circuit processing and optimization. Here's an overview of the workflow:

Analysis of Qiskit's Default Transpiler:

At this stage, the system analyzes Qiskit's default transpiler. It involves transpiling quantum circuits of various sizes and complexities across different optimization levels offered by Qiskit. The goal here is to understand the performance of the transpiler in terms of its runtime efficiency and the quality of the output circuits.

Qiskit's Transpilation Process Breakdown:

This part of the workflow delves into the detailed aspects of Qiskit's transpilation process. It breaks down the transpiler into its passes that follows each other, analyzing the time and resource efficiency of each pass. The emphasis is on optimization level 3, the most intensive level, to understand how the transpiler behaves under maximal optimization demands.

Analysis of State-of-Art Qubit Mapping and Routing Algorithms:

This phase evaluates various mapping and routing algorithms crucial for qubit layout optimization in quantum circuits. It includes both heuristic and constraint-based approaches, with a focus on algorithms like SMT, MAXSAT, and SABRE. The analysis aims to assess the algorithms' effectiveness in optimizing circuit layouts and routing, with attention to factors like swap gate minimization and overall circuit fidelity.

Transpilation with a Sub-Architecture:

This aspect of the system focuses on the strategic selection of quantum subarchitecture for executing quantum circuits. The objective is to match the circuit's requirements with the most suitable quantum hardware, considering factors such as qubit count, connectivity, and fidelity. This optimization is achieved through a detailed analysis of the backends' performance metrics and the implementation of an intelligent backend selection logic, ensuring that each circuit is run on the optimal hardware available, thus maximizing runtime efficiency and fidelity.

3.3 System Components

The system comprises the following key components:

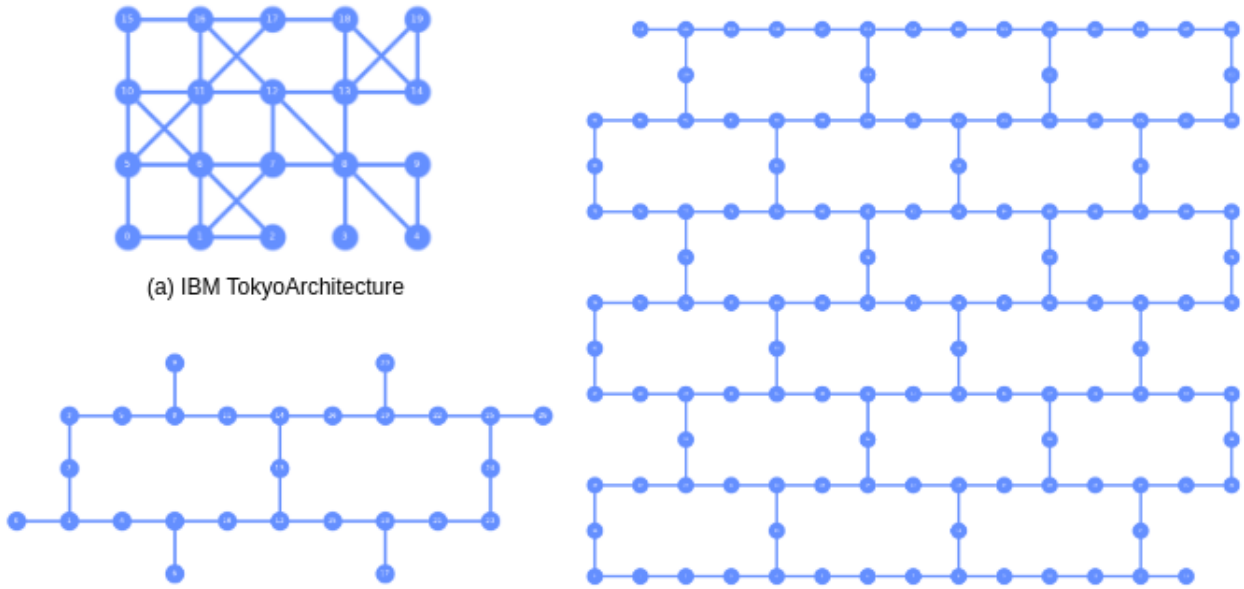
Qiskit Transpiler: A core component used for the compilation and optimization of quantum circuits. It offers different optimization levels and is crucial for adapting circuits to specific quantum hardware.

Mapping and Routing Algorithms: These algorithms, like SMT, MAXSAT, and SABRE, are integral for optimizing the layout and routing of qubits in the quantum circuits.

Quantum Hardware Backends: Various IBM Quantum backends (such as Washington, Tokyo, and Kolkata) are used to test and validate the circuits. Each backend has its unique qubit count and connectivity, influencing the circuit's performance.

Performance Analysis Tools: Tools and metrics are used to evaluate the efficiency and quality of the transpiled quantum circuits.

Backend Selection Logic: This component is responsible for choosing the most suitable quantum backend or subbackend based on the circuit's requirements and the backend's characteristics.



(a) IBM TokyoArchitecture

(b) IBM Kolkata Architecture

(c) IBM Washington Architecture

Figure 3.1: Various IBM Quantum System's Architectures

4 Design

4.1 Analysis of Qiskit's Transpiler

The initial experiments with Qiskit's default transpiler involved assessing its performance across various optimization levels and circuit sizes. The transpiler in Qiskit offers different optimization levels, ranging from 0 (no optimization) to 3 (highest level of optimization). Each level provides a balance between compilation time and the efficiency of the resulting circuit.

Experiment 1: Optimization Level Comparison - Testing circuits of varying sizes (number of qubits) across all optimization levels to evaluate the transpiler's performance in terms of runtime and output circuit quality.

Experiment 2: Maximum Circuit Size Analysis - To understand the limitations of the transpiler by progressively increasing the size of the quantum circuits and attempting to compile them using optimization levels 2 and 3. This experiment aimed to identify the largest circuit size that could be feasibly compiled on a standard computing setup.

Experiment 3: Backend-Specific Transpilation Analysis - Evaluating the performance of the transpiler across three different quantum backends (Washington, Tokyo, Kolkata) with varying numbers of qubits, while using optimization level 3. This experiment was designed to explore how different quantum hardware architectures impact the transpiler's runtime efficiency and output circuit quality. It involved transpiling random circuits with an increasing number of qubits and measuring the average runtime for each backend. Figure 3.1 shows the specifics of the backends used in this study. Notably, IBM Washington has the highest qubit count with 127 qubits, followed by IBM Kolkata with 27 qubits, and IBM Tokyo with 20 qubits. A critical aspect of this comparison is the connectivity of these backends, with IBM Tokyo exhibiting the highest level of qubit connectivity. This factor is anticipated to significantly influence the quality of the output circuits, especially in terms of the number of swap gates required.

4.2 Qiskit's Transpilation Process Breakdown

In this section, a detailed analysis of Qiskit's transpilation process, particularly focusing on the breakdown of its passes and their performance is conducted. The objective is to understand which passes of the transpilation process are the most time-intensive and how they scale with different circuit sizes. This analysis is crucial for optimizing the transpilation process and improving overall efficiency.

Qiskit's transpiler is composed of several passes, each handling a specific aspect of the transpilation process. By utilizing a diverse set of quantum circuits, varying in size and complexity, the performance of each transpilation pass is assesed. The focus was on circuits transpiled at optimization level 3, Qiskit's highest level of optimization, which offers the most comprehensive optimization but also demands the most computational resources.

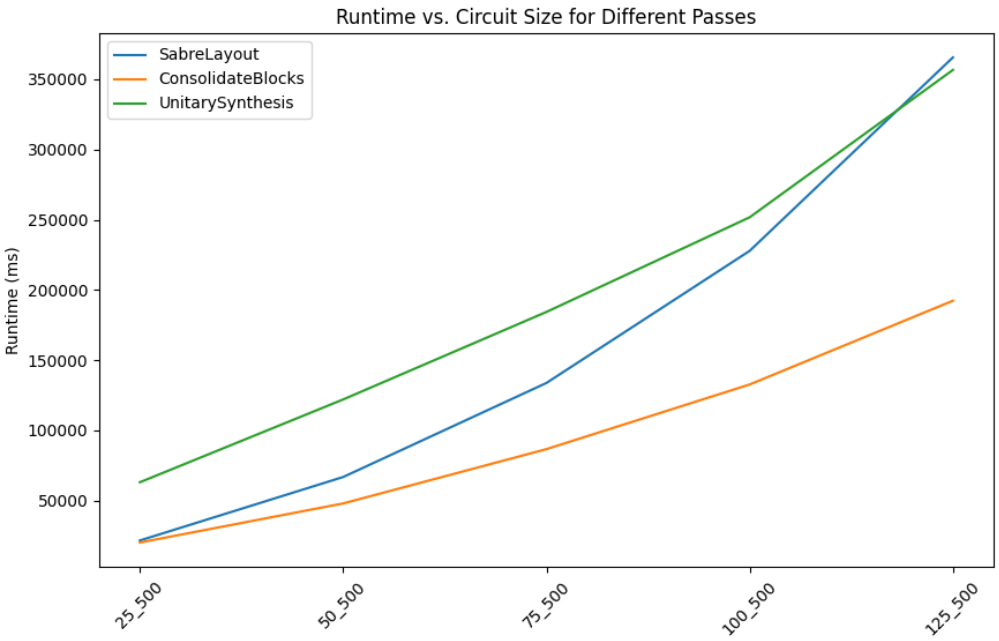


Figure 4.1: Runtime of passes vs Circuit Size. x axes shows the circuit size that is transpiled in terms of <number-of-qubits>_<depth>

The results depicted in Figure 4.1 highlight three transpiler passes with the most significant runtime in Qiskit's entire transpilation process: ConsolidateBlocks, UnitarySynthesis, and SabreLayout.

UnitarySynthesis: This pass synthesizes unitaries over certain basis gates. It can approximate 2-qubit unitaries, leveraging gate fidelities either provided through backend_props or target. The approximation degree can be adjusted via the approximation_degree heuristic dial.

ConsolidateBlocks: This pass consolidates sequences of uninterrupted gates acting on the same qubits into a single Unitary node. The goal is to replace these sequences with a more optimal subcircuit upon later resynthesis.

SabreLayout: A powerful algorithm for quantum mapping and routing. As the circuit size increases, this pass exhibits the most significant increase in runtime, prompting further investigations into its efficiency and performance. This also shows that Qiskit uses Sabre Layout as the default mapping algorithm.

Given SabreLayout's substantial runtime, especially with larger circuits, next analysis are designed to investigate this pass in greater depth also try and compare new quantum mapping and routing algorithms.

4.3 Analysis of State-of-Art Qubit Mapping and Routing Algorithms

This section delves into analyzing practical performance of State-of-Art qubit mapping and routing algorithms in quantum computing. Complementing the theoretical framework, experiments conducted to evaluate the runtime efficiency and the quality of the circuits generated by each algorithm. These experiments encompass a variety of quantum circuits with differing sizes. In this exploration, these algorithms are categorized into two primary groups: heuristic and constraint-based approaches.

4.3.1 Constraint-based Approaches

Constraint-based methods involve defining a set of constraints that the solution must satisfy. These methods are designed to find the best possible solution within the defined constraints, which often means they can guarantee optimality. These methods can be computationally demanding, as they may require exploring a large solution space to find the optimal solution. In context of qubit mapping and routing, these solutions try every possible mapping for targeted quantum computer and return the solution with least amount of swap gates added. For larger circuits, constraint-based methods may become less practical due to their computational demands.

[8], [17], [19], [9], [11] employ constraint-based approaches. These methodologies, while distinct in their implementation, share common constraints to ensure the accurate and efficient execution of quantum circuits.

Common Constraints Across All Methods:

- **Unique Assignment of Logical to Physical Qubits:** Each logical qubit must be assigned to a unique physical qubit. You can't have two different logical qubits sharing the same physical qubit simultaneously.
- **Preservation of Gate Execution Order:** The original sequence of gate operations in the circuit must be maintained in the final mapped version. This fidelity to the original circuit design is crucial for ensuring that the computation performed by the quantum circuit remains accurate and true to its intended function.
- **Adjacency Requirement for Two-Qubit Gates:** Whenever a gate operates on two qubits, those qubits must be placed on neighboring physical qubits in the quantum processor. This constraint is due to the physical limitations and connectivity of the qubits in quantum hardware.
- **Use of SWAP Gates for Non-Adjacent Qubits:** If a two-qubit gate involves qubits that are not adjacent on the processor, This rearrangement ensures that the necessary qubits are adjacent when needed for a gate operation.
- **Considerate Insertion of SWAP Gates:** Inserting SWAP gates must be done carefully to avoid interference with the execution of other gates. This requires strategic planning of when and where to insert SWAP operations to avoid conflicts.

All these constraint-based approaches for qubit mapping and routing in quantum computing share the common objective of reducing the number of SWAP operations, though the terminology and method of incorporation into the optimization process may vary among different techniques.

Satisfiability Modulo Theories (SMT)

[8], [17], [19] employs SMT which extends Boolean satisfiability problem (SAT). In SMT-based approaches, minimizing the number of SWAP operations is typically integrated into the cost function or objective function of the problem. The SMT solver works to find a solution that not only satisfies all the constraints (like mapping and adjacency requirements) but also minimizes this cost function, which represents the number of SWAPs.

[17] introduces Optimal Layout Synthesis using Quantum Computing (OLSQ) which uses Z3 Solver [3]. However, it faces challenges with scalability and complexity, especially for larger quantum circuits. To address this, Transition-Based OLSQ (TB-OLSQ) is developed as a variant of OLSQ. It schedules gates into blocks between mapping

transitions rather than individually, with no SWAP gates inside a block. This approach reduces the problem's complexity but may lead to sub-optimal solutions in terms of SWAP gate minimization.

Further advancing these concepts, **TB-OLSQ2**, as introduced in [8], builds on the foundation of TB-OLSQ. It uses a coarse-grained circuit model similar to TB-OLSQ but implements a more efficient optimization strategy and variable encoding. The transition-based model in TB-OLSQ2 allows it to handle larger quantum circuits and more complex layouts than OLSQ and TB-OLSQ hence this study has chosen TB-OLSQ2 for the comparision.

Maximum Satisfiability (MAXSAT)

The study in [9] utilizes MAXSAT, an optimized form of the SAT problem, for quantum circuit mapping. MAXSAT approaches the minimization of SWAP gates as a "soft constraint," balancing it with the necessity of meeting the "hard constraints" for a valid solution. The method ensures that while minimizing the number of SWAP gates, the hard constraints of the solution are not compromised.

Additionally, [9] introduces a technique of local relaxation by segmenting the quantum circuit into sequential slices. Each slice is optimized individually, ensuring local optimality in terms of the least number of swaps. However, this focus on local optimality for each slice doesn't guarantee global optimality for the entire circuit. The whole approach is implemented in a tool called **satmap**[1].

Binary Integer Program (BIP)

[11] employs BIP and uses IBM CPLEX[2] as their BIP Solver. Similarly, The primary goal of this approach is to find a circuit implementation that maximizes the circuit fidelity while minimizing the use of quantum resources.

4.3.2 Heuristic Approaches

Heuristic methods rely on educated guesses to find solutions. They do not guarantee the best solution but aim to find a good enough solution in a reasonable amount of time. These methods are generally faster and can provide solutions more quickly than constraint-based methods. This makes them suitable for large-scale problems or situations where a quick solution is more valuable than an optimal one. They strike a balance between solution quality and computational efficiency.

[1] https://github.com/qqq-wisc/satmap
[2] https://www.ibm.com/products/ilog-cplex-optimization-studio

SWAP-based Bidirectional heuristic search algorithm (SABRE)

SABRE developed by Li et al. in their work [7], represents a significant advancement in the field. SABRE employs a unique bidirectional search strategy. This technique involves simultaneously exploring mappings from both the beginning and end of the quantum circuit. This bidirectional approach allows SABRE to more effectively navigate the solution space and find efficient mappings, especially in scenarios where the circuit's beginning and end have different qubit requirements. Central to SABRE's approach is its use of heuristics to guide the search process. These heuristics are designed to make educated decisions about which qubits to swap at each step of the process, aiming to reduce the overall number of swaps needed. SABRE processes the quantum circuit layer by layer. It examines each layer of gates in the circuit and applies its mapping and swapping heuristics to ensure that the qubits involved in each gate are adjacent on the physical qubit graph. This layer-wise processing helps in managing the complexity of the circuit and ensures that the mapping remains efficient throughout the computation. A key goal of SABRE is to minimize the number of SWAP gates introduced into the circuit. SWAP gates are necessary to bring non-adjacent qubits together for multi-qubit gates, but they also add to the circuit's depth and complexity. SABRE's heuristics are tailored to find solutions that keep the SWAP count as low as possible. SABRE is designed to be adaptable to different quantum hardware architectures. It can be applied to processors with various qubit connectivity patterns, making it a versatile tool for a wide range of quantum computing platforms. While it may not always find the optimal solution, it provides a balance between solution quality and computational resources. SABRE is implemented in Qiskit.

Noise-Adaptive

The method[10] utilizes real-time calibration data, including error rates and gate times, to inform qubit mappings and gate schedules, optimizing hardware performance. Focuses on maximizing the program's reliability by placing qubits at low-error-rate hardware locations such as CNOT errors, readout errors, and minimizing qubit movement to reduce errors. Implements strategies like "Rectangle Reservation" and "One Bend Paths" for CNOT gate routing, essential for execution duration and reliability. Alongside optimization methods, heuristics like "Greedy Vertex First" and "Greedy Edge First" balance reliability optimization and scalability for larger systems. This noise-adaptive approach is implemented in Qiskit.

Time-Optimal

The paper [21] introduces a Time-Optimal Qubit Mapping (TOQM) scheme for efficiently mapping quantum circuits onto IBM's quantum hardware. The primary focus is on reducing the overall execution time of the quantum circuit, rather than solely minimizing the number of additional SWAP gates introduced during the mapping process. This time-optimization approach involves creating a heuristic function based on the dependency graph of the remaining circuit and the current qubit mapping. The heuristic function calculates the minimal time required to execute the remaining circuit, taking into account both the direct and indirect predecessors of each gate and the potential delays caused by inserting SWAP gates. The heuristic function ensures that the estimated execution time of the remaining circuit never exceeds the actual time, thereby providing a lower bound on the execution time.

QMAP

In the study [22], an algorithm is presented for mapping logical qubits of quantum circuits to the physical qubits of IBM's QX architecture. This process involves partitioning the quantum circuit into layers. Each layer consists of gates that operate on different qubit sets, allowing simultaneous gate execution within a layer. This layer-based approach ensures that CNOT-constraints are met for all gates in a layer, reducing the frequency of remapping. The A*-search algorithm then finds a CNOT-constraint-compliant mapping for each layer, transitioning from the mapping of the previous layer by adding SWAP operations. The objective is to minimize additional operations by keeping the new mapping close to the previous one. The algorithm employs heuristics based on the shortest paths in the IBM QX architecture's coupling map, providing an accurate estimation of the distance to a compliant mapping without overestimating costs.

This methodology is implemented in MQT QMAP - A tool for Quantum Circuit Compilation [3], It is referred to as **QMap-Heuristic** in this study. Same tool also includes a constraint-based approach [19], although it is not part of this analysis.

4.3.3 Comparative Analysis of Mapping and Routing Algorithms

This part outlines the methodology for our comparative analysis of various qubit mapping and routing algorithms. The comparison involved both the methods natively available in Qiskit and several external algorithms, each offering unique approaches to quantum circuit transpilation.

[3] https://github.com/cda-tum/mqt-qmap

For this comparative study, a range of algorithms was selected, including both Qiskit's native methods and external ones:

- **Qiskit's Algorithms** Dense, Trivial, Dense, and Sabre.
- **External Algorithms:** TOQM[21], SATMap[9], TB-OLSQ2[8], and QMap-Heuristic[22].

The comparative analysis was conducted in two phases:

1. **Small Circuit Batch:** Initially, a sample batch of smaller circuits was selected for comparison. This phase primarily focused on evaluating constraint-based methods, which are generally less efficient at compiling larger circuits or may require extensive computational time (exceeding 24 hours in some cases). This phase provided insights into the algorithms' performance in relatively simpler scenarios.
2. **Large Circuit Batch:** In the second phase, the analysis was extended to larger circuits. This phase included the comparison of Qiskit's native methods (Dense, Trivial, Dense, Sabre) with external algorithms like TOQM and QMap Heuristic, known for their efficiency in handling larger quantum circuits.

The algorithms were evaluated based on three key performance indicators:

- **Compilation Time:** The time taken to compile quantum circuits of varying complexities.
- **Circuit Quality:** Assessing the quality of the output circuits, particularly focusing on the number of SWAP gates introduced and the overall circuit depth.
- **Scalability:** How well the algorithm performs as the size and complexity of the quantum circuit increase.

4.4 Transpilation with a Subarchitecture

Results from Section 4.1 experiment 3 showed that employing a smaller backend in transpilation process can yield a similar quality circuit but in a shorter time frame. So this section focuses on the strategic selection of quantum subarchitecture to optimize the execution of quantum circuits in terms of runtime.

4.4.1 Subarchitecture Selection for IBM Kolkata

For IBM Kolkata, the process begins by identifying sub-architectures, each characterized by a progressively smaller number of qubits (e.g., 27, 24, 21, 18, and so on). These subarchitectures are derived from the backend's qubit coupling graph, which is a fixed attribute of the backend.

For each subarchitecture a fidelity score is calculated. Calculation begins with fetching the calibration data from IBM Kolkata backend. The next step involves calculating a 'fidelity score' for each subarchitecture. A higher score indicates a higher overall fidelity. This score is computed by considering each qubit's readout and single-qubit gate errors and each link's (CNOT gate) error within the subgraph. The computation method follows the approach used in Mapomatic[12], which adjusts fidelity based on individual qubit and link errors.

For a given quantum circuit, the sub-architecture closest in qubit count and with the highest fidelity score is chosen. The effectiveness of this selected sub-architecture is then evaluated using two different layout algorithms: the SABRE layout algorithm (Qiskit's default) and the constraint-based method Satmap.

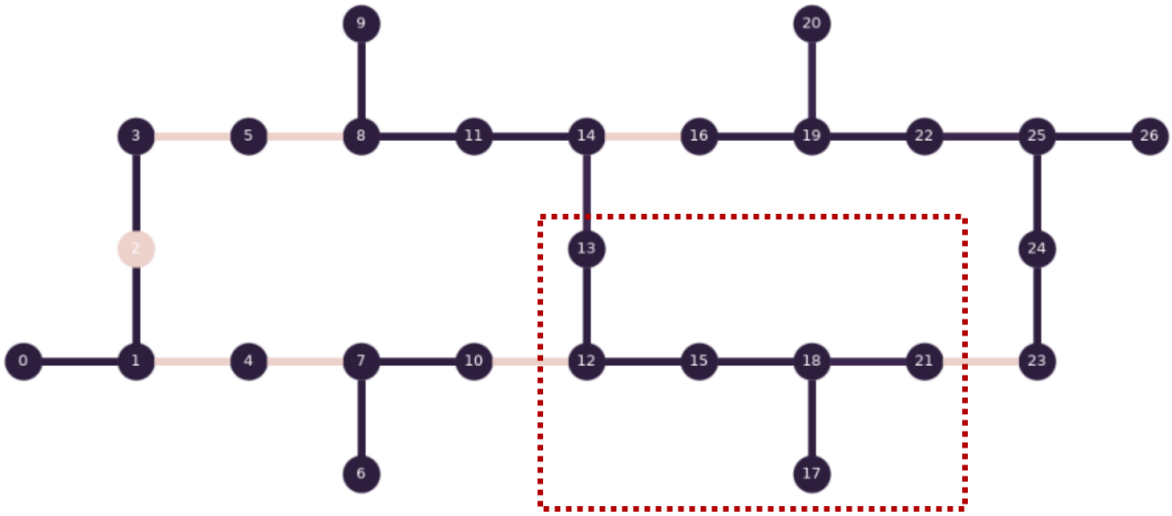


Figure 4.2: Example Subarchitecture with 6 Qubits Selection for IBM Kolkata

4.4.2 Subarchitecture Selection for IBM Brisbane

IBM Brisbane, with its extensive 127-qubit architecture mirrors that of IBM Washington(Figure 3.1). Since Section 4.1 experiment 3, showed that transpiling a quantum circuit smaller than 20 qubits with IBM Kolkata is faster than transpiling it with IBM Washington, a method is designed to extract 27-qubit sub-architectures from IBM Brisbane using a SAT solver. This solver identifies the top 10 sub-architectures based on number of edges, ensuring the selection of highly interconnected qubit networks for efficient transpilation. Among these 10, the sub-architecture with the highest fidelity—calculated using the same criteria as in Subsection 4.4.1—is selected. The transpilation efficiency of this selected sub-architecture is then assessed using the SABRE layout algorithm.

5 Implementation

5.1 Analysis of Qiskit's Transpiler

5.1.1 Experiment 1: Optimization Level Comparison

In Experiment 1, the goal was to evaluate the performance of Qiskit's transpiler across its various optimization levels. To achieve this, quantum circuits of different sizes were created and then transpiled at different optimization levels. The quantum circuits were generated using Qiskit's `random_circuit(num_qubits, circuit_depth)` function. This function creates random quantum circuits with a specified number of qubits (num_qubits) and circuit depth (circuit_depth). Each generated circuit was transpiled using Qiskit's `transpile(circ, backend, optimization_level=opt_level)` function. This function converts the high-level circuit into a lower-level representation that can be executed on a quantum device. The optimization_level parameter, varying from 0 to 3, indicates the degree of optimization applied during transpilation. The backend used for the transpilation was FakeTokyo(). In Qiskit, a fake backend is a simulated version of an actual quantum device. These fake backends mimic the characteristics (like qubit connectivity and error rates) of real quantum devices without actually executing the circuits. Since experiment's aim is to investigate the transpilation process, and not running the circuit on an actual quantum backend, using a fake backend provided a consistent and controlled environment. For each combination of circuit size and optimization level, the time taken by the transpiler to process the circuit was recorded.

5.1.2 Experiment 2: Maximum Circuit Size Analysis

Experiment 2 aimed to explore the limits of Qiskit's transpiler by progressively increasing the size of quantum circuits. The implementation involved creating random quantum circuits with a gradually increasing number of qubits from 5 to 125 and circuit depth from 50 to 1051. Each circuit was transpiled using the FakeWashingtonV2 backend at optimization level 2. The execution time for each transpilation process was recorded to evaluate performance scalability. The same experiment was also done with optimization level 3.

5.1.3 Experiment 3: Backend-Specific Transpilation Analysis

The third experiment was designed to evaluate the performance of Qiskit's transpiler across different quantum hardware backends. In this experiment, a series of random quantum circuits of varying sizes were transpiled using three different backends: FakeWashingtonV2, FakeTokyo, and FakeKolkata. These backends were selected to represent a range of qubit counts and connectivity. For each circuit and backend, the time taken to transpile and the number of swap gates in the resulting circuits were recorded. To count the number of swap gates added, a function was implemented that iterates through the transpiled circuit's instructions. It identifies sequences of CNOT gates that collectively represent a swap operation and tracks their occurrence.

5.2 Qiskit's Transpilation Process Breakdown

The implementation involved creating a series of random circuits of increasing sizes in terms of number of qubits. Each of these circuits was then subjected to the transpilation process using Qiskit's `transpile(circ, backend, optimization_level=3, callback=step_collector.on_step_callback)` function. The callback parameter in the transpile function is called after each pass execution during the transpilation process. This callback function receives several keyword arguments that reveal the internals of each pass in the transpilation, such as the time to execute the pass. Customized on_step_callback function is designed handle the information received from the transpiler's callback mechanism and to store it in a CSV file.

5.3 Analysis of State-of-Art Qubit Mapping and Routing Algorithms

5.3.1 Qiskit's Algorithms

The test involved transpiling quantum circuits using Qiskit's 4 different layout methods which are 'trivial', 'dense', 'noise_adaptive', and 'sabre'. `transpile(circ, backend, layout_method=layout, optimization_level=3)` These algorithms were applied to a 2 set of quantum benchmark circuits stored in QASM files. In the first set there were smaller circuits, size varying from 5 to 480. The second set included bigger circuits size varying from 200 to 184864. Here size refers to number of gates the circuit have. Both benchmark sets are taken from [21].

For each file, a QuantumCircuit object was created using Qiskit's QuantumCircuit.from_qasm_file method. The number of qubits and the depth of each circuit were

recorded, and the circuits were transpiled with each of the Qiskit methods using the transpile function. The FakeTokyo backend which has 20 qubits was employed as a testbed for the transpilation process. The execution time for transpiling each circuit was measured, and the number of swap gates added during the transpilation process was counted, and the results were recorded.

5.3.2 External Algorithms

In addition to Qiskit's native methods, this study also examined four external mapping and routing algorithms: TOQM, SATMap, TB-OLSQ2, and QMap Heuristic. Their performance was tested using the same set of large and small quantum circuits, and their execution times, as well as the swap gate counts, were similarly recorded for a comparative evaluation. Unlike Qiskit's integrated methods, these external algorithms required individual execution of their source codes. Here are the specific commands and methodologies used to run each algorithm:

- **TB-OLSQ2:** The TB-OLSQ2 algorithm was executed using a Python script. The command used is as follows:

```
python run_olsq.py --dt 'tokyo' --qf qasm_file
    --f ./outputs --swap --sabre --tran
```

- **SATMap:** The SATMap algorithm was run with the following Python configuration:

```
satmap.MappingArguments(
    prog=qasm_file, arch='tokyo', output_path=output,
    timeout=1800, k=25, cyclic='off', no_route=False,
    weighted=False, err=None)
satmap.map_program(args)
```

- **TOQM:** TOQM was executed through a command line interface, with the following command:

```
./mapper qasm_file couplings/tokyo.txt -defaults
    -latency Latency_1_2_6 -expander GreedyTopK 10
    -queue TrimSlowNodes 2000 1000 -nodeMod GreedyMapper -retain 1
```

- **QMap Heuristic:** The QMap Heuristic algorithm was run with the following Python:

```
circ_mapped, results = qmap.compile(circ, arch=FakeTokyo(),
    method="heuristic", post_mapping_optimizations=False)
```

As seen in the implementation same Tokyo architecture is used for every external algorithm to ensure consistency with Qiskit's methods.

5.4 Transpilation with a Subarchitecture

5.4.1 Calibration Data Collection

The optimization of quantum backend usage began with a comprehensive collection of calibration data for the real Kolkata and Brisbane backends. The data was fetched using the IBMProvider interface in Qiskit. The properties of each backend were collected over the last six months and stored in a JSON files for each days. This long-term data collection allowed for averaging the performance metrics over different periods: the last week, the last month, and the last six months. This averaging process provided a more accurate and robust understanding of each backend's performance over time.

5.4.2 Subarchitecture Extraction

For the Kolkata backend, subarchitectures were extracted in decreasing steps of qubit counts, such as 27, 24, 21, 18, etc. This was achieved using the NetworkX library in Python, specifically utilizing the

```
subgraphs = [graph.subgraph(nodes) for nodes in combinations(graph.nodes(),size)
    if graph.subgraph(nodes).number_of_nodes() == size and
    nx.is_connected(graph.subgraph(nodes))]
```

function to identify subgraphs within the larger qubit connectivity graph. These subgraphs were then stored for further analysis.

For Brisbane and other 127-qubit backends, a different approach was required due to runtime constraints. The mathematical solver called LocalSolver was employed to find the 10 most connected subgraphs for a given number of qubits.

5.4.3 Fidelity Calculation

Once the subarchitectures were identified, the fidelity of each subgraph was calculated. This involved multiplying the individual gate and readout error rates of each qubit and the error rates of the CNOT gates between qubits in the subgraph. The fidelity calculation was performed as follows:

```
for node in nodes:
    fid *= 1 - props.gate_error('sx', node)
    fid *= 1 - props.gate_error('x', node)
    fid *= 1 - props.readout_error(node)
edges = find_subgraph_edges(cmap, nodes)
for (q1, q2) in edges:
    fid *= (1 - props.gate_error('cx', [q1, q2]))
```

This approach to fidelity calculation is similar to that used by Mapomatic [12].

5.4.4 Subgraph Selection and Transpilation

The subgraph with the highest fidelity was selected for use in the transpilation process. Transpilation was performed using the `transpile(circ, coupling_map=cmap, optimization_level=3)` function in Qiskit, with the coupling map (cmap) of the chosen subgraph as an argument instead of the entire backend. The method uses the SABRE layout by default. An additional experiment was conducted using the SATMap algorithm, where the selected subgraph, instead of the entire backend, was provided as the architecture for transpilation.

6 Evaluation

Table 6.1 presents information regarding the system used for all evaluations in this chapter.

Property	Value
CPU Model	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
Architecture	x86_64
CPU(s)	12
CPU Threads per Core	2
System Memory	7.6G
Operating System	Ubuntu 18.04.6 LTS
Linux Kernel	5.4.0-150-generic

Table 6.1: Information of system used in these evaluations

6.1 Evaluation of Qiskit's Transpiler

6.1.1 Experiment 1: Optimization Level Comparison

The first experiment conducted using Qiskit's transpiler on a set of quantum circuits with varying numbers of qubits and a fixed circuit depth of 500. The optimization levels tested range from 0 to 3, with level 0 representing no optimization and level 3 representing the highest level of optimization using IBM Tokyo Backend.

Figure 6.1 present the average transpilation runtime as a function of the number of qubits for different optimization levels. The data suggests that the runtime increases with the number of qubits for all optimization levels. However, the rate of increase is steeper for higher optimization levels, indicating that the computational complexity of optimization grows with the size of the quantum circuit.

Table 6.2 shows the average number of swap gates added during transpilation for quantum circuits with varying numbers of qubits across different optimization levels. Notably, as the optimization level increases, the number of swap gates tends to decrease.

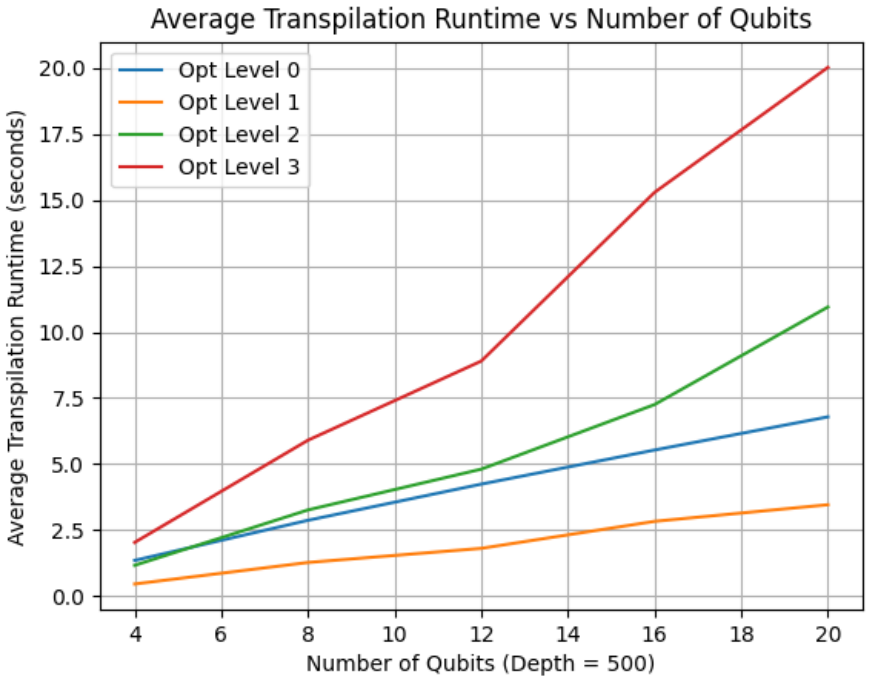


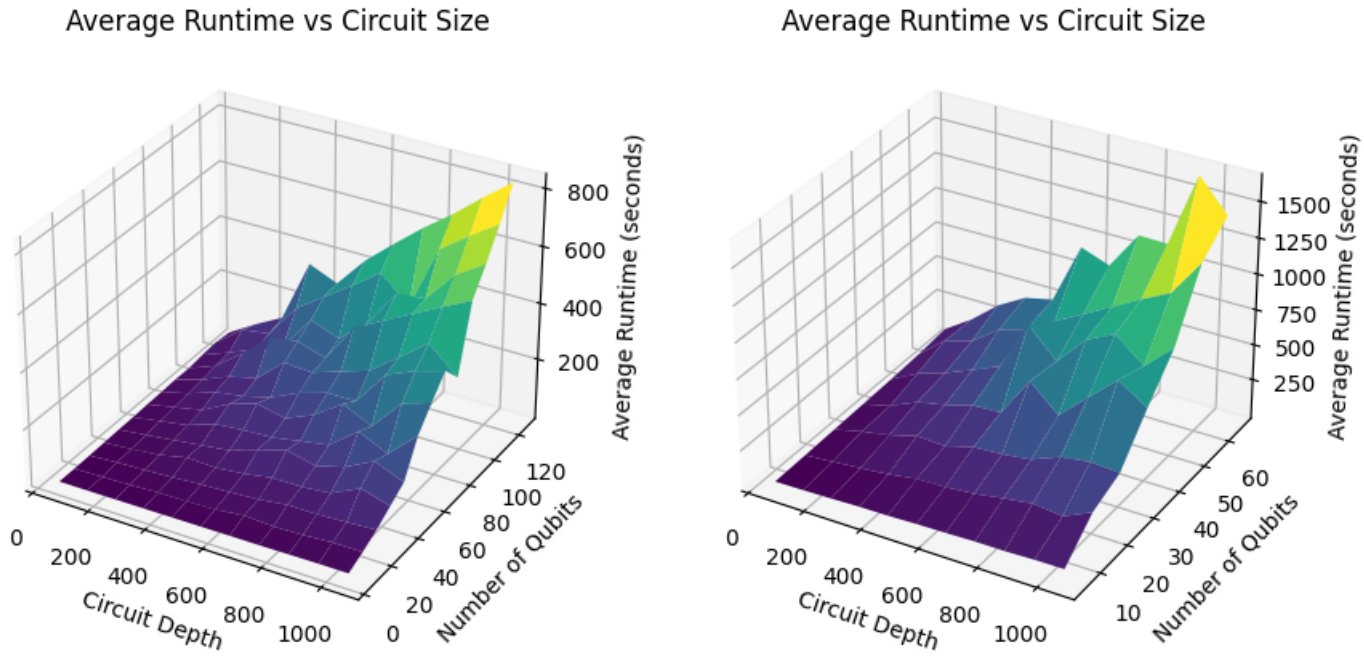
Figure 6.1: Average Transpilation Runtime vs Number of Qubits for Different Optimization Levels

Number of Qubits	Depth	Opt Level 0	Opt Level 1	Opt Level 2	Opt Level 3
4	500	278.33	22.33	19.33	10.00
8	500	1683.00	770.67	751.00	605.00
12	500	3661.67	1677.33	1560.00	1372.67
16	500	5992.33	2845.67	2835.33	2489.00
20	500	8728.33	4127.67	4155.00	3763.33

Table 6.2: Average Swap Gates Added to Each Circuit After Transpilation Process for Different Optimization Levels

6.1.2 Experiment 2: Maximum Circuit Size Analysis

The second experiment delves into the scalability of the transpilation process in quantum computing, assessing the maximum circuit size that can be feasibly compiled with Qiskit's transpiler on a standard computing setup. The experiment involves transpiling quantum circuits with an increasing number of qubits and circuit depth, employing optimization levels 2 and 3 to examine the transpiler's limits.



(a) Optimization Level 2

(b) Optimization Level 3

Figure 6.2: Average Runtime of Transpilation Process vs Circuit Size (Depth and Qubit Size) for Different Optimization Levels

When using optimization level 3, the transpiler reached a computational bottleneck; the system used in this experiment run out of memory while attempting to compile circuits exceeding 60 qubits and a depth of 500. This limitation is a significant concern for the scalability of quantum computations, especially considering the extensive qubit requirements for algorithms like Shor's, which needs millions of qubits for practical applications[18].

Furthermore, Figure 6.2 shows that the runtime for transpiling a quantum circuit with 100 qubits and a depth of 1000 using optimization level 2 is around 10 minutes. This duration, when contrasted with classical computing standards, is substantial and points to an inefficiency in the current state of quantum computing transpilation.

The experimental results underscore a critical challenge in the field of quantum computing: the need for enhanced transpilation processes that can manage larger quantum circuits within practical time frames and computational resources. As the quantum computing field advances towards solving more complex problems, the transpilation runtime and resource consumption become increasingly pivotal factors.

6.1.3 Experiment 3: Backend-Specific Transpilation Analysis

The experimental testbed comprises three IBM quantum backends—Washington, Tokyo, and Kolkata—with varying qubit connectivity. Tokyo stands out with the highest connectivity, followed by Kolkata, which is a subset of Washington's architecture as shown in Figure 3.1. The methodology involves transpiling quantum circuits with a fixed depth of 500 and an increasing number of qubits (4, 8, 12, 16, and 20) on each backend.

Theoretically, backends with higher connectivity should require fewer swap gates due to the increased availability of direct qubit interactions, potentially resulting in shorter transpilation times.

Figure 6.3 and Table 6.3 showcases that as the connectivity of the quantum hardware increases, the transpilation runtime and the number of swap gates added tend to decrease. IBM Tokyo, with its superior connectivity, consistently shows the lowest runtime and fewest swap gates, affirming the notion that highly connected backends can lead to more efficient quantum computation.

When contrasting the performance on IBM Kolkata against IBM Washington, there is a notable reduction in the transpilation runtime for Kolkata, despite it being a subgraph of Washington. Surprisingly, the swap gate count remains nearly unchanged, suggesting that employing a subarchitecture of a larger backend can yield a similar quality circuit but in a shorter time frame.

IBM Washington's relatively sparse connectivity leads to higher runtimes and more swap gates. Conversely, IBM Tokyo's denser connectivity pattern results in superior transpilation performance, emphasizing the importance of choosing the right backend for quantum circuit execution.

The evaluation underscores the importance of backend selection in quantum circuit transpilation. A backend with better connectivity not only enhances transpilation efficiency but also simplifies the resulting circuit, which is pivotal for the practicality of quantum computations.

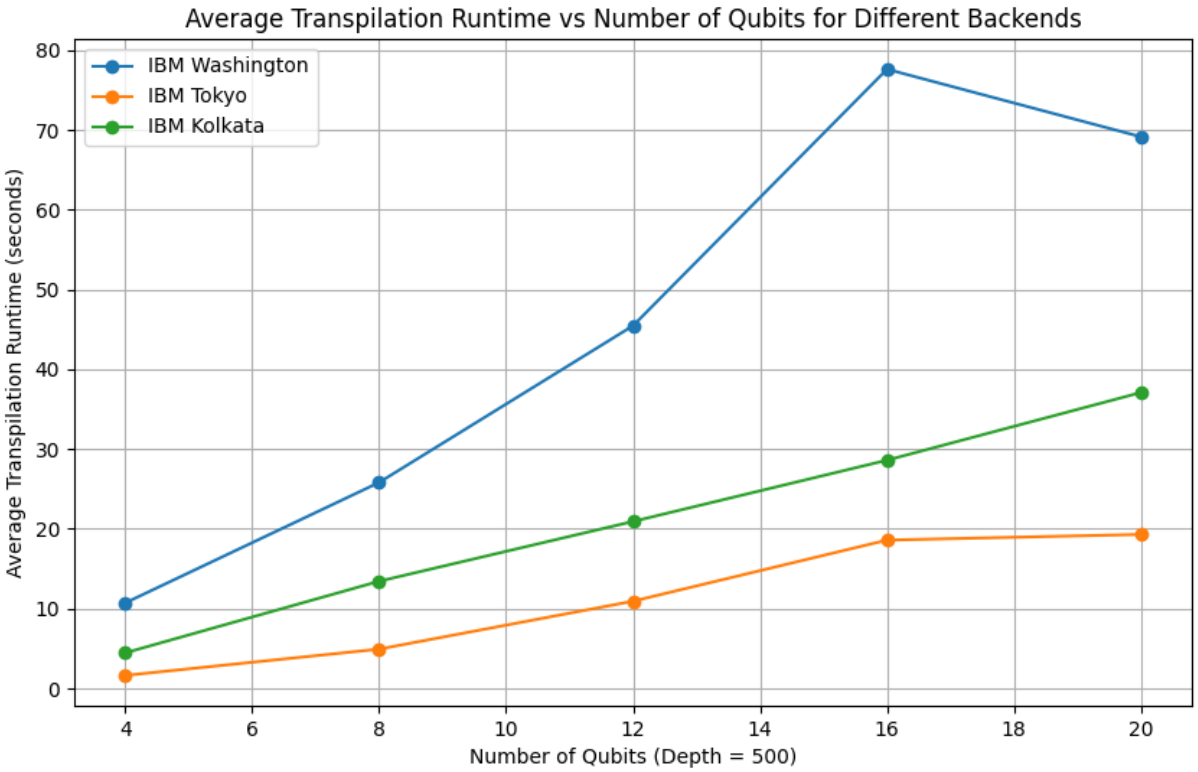


Figure 6.3: Average Transpilation Runtime vs Number of Qubits for Different Backends

Number of Qubits	Depth	IBM Washington	IBM Tokyo	IBM Kolkata
4	500	279	12	270
8	500	1441	686	1494
12	500	3169	1375	3215
16	500	5372	2528	5553
20	500	8822	3753	8791

Table 6.3: Swap Gates Added to Each Circuit for Different Backends After Transpilation process

6.2 Evaluation of Different Quantum Mapping and Routing Algorithms

The evaluation compares different quantum mapping and routing algorithms, ranging from Qiskit's native methods trivial, dense, noise_adaptive, and SABRE to external ones TOQM, SATMap, TB-OLSQ2, and QMap. These algorithms were evaluated based on their runtime efficiency and the quality of the transpiled circuits, particularly focusing on the number of SWAP gates introduced. IBM Tokyo Backend architecture used for analyzing all the methods. The comparative analysis was conducted in two phases first with small sized quantum circuits, then with larger quantum circuits.

6.2.1 Evaluation with Small Circuit Batch

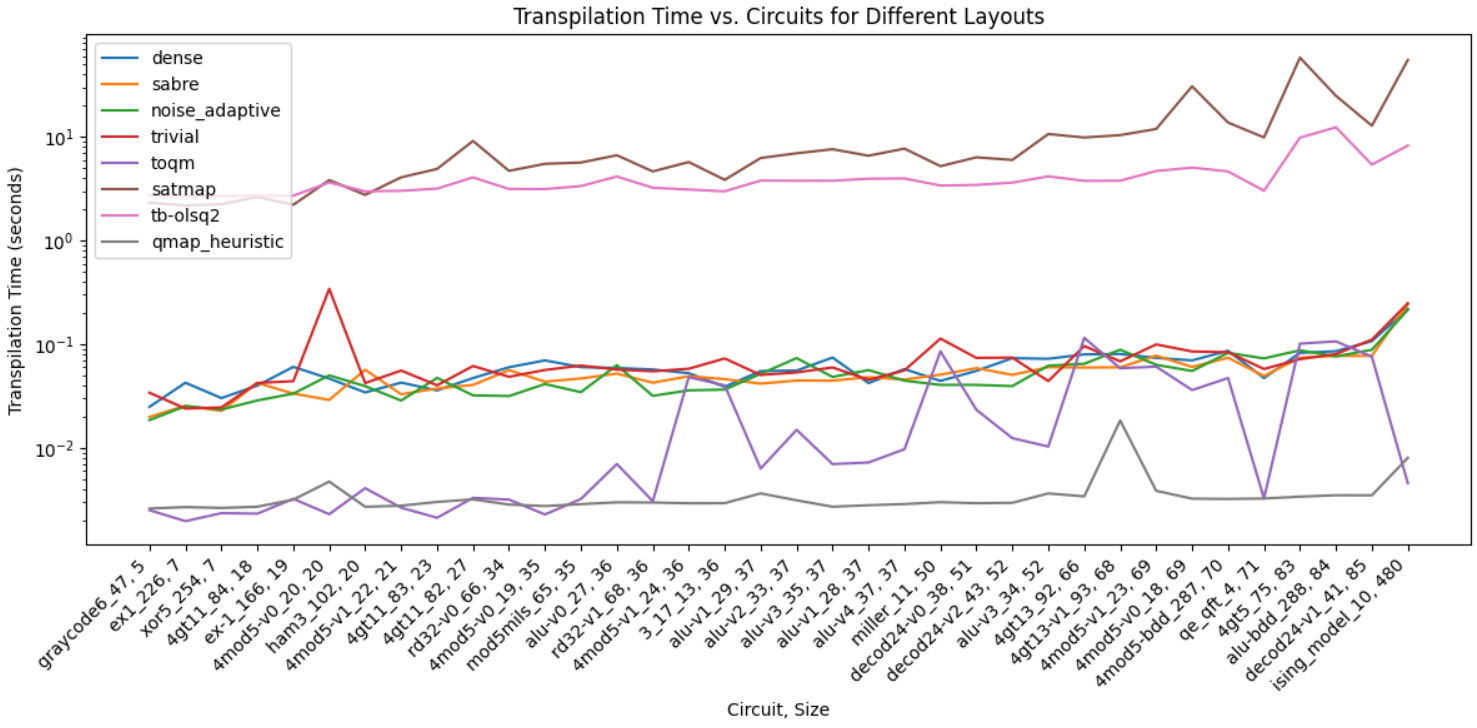


Figure 6.4: Runtime of Different Mapping and Routing Algorithms vs Smaller Quantum Circuits

Figure 6.4 shows runtime of different mapping and routing algorithms for smaller benchmark quantum circuits sizes ranging from 5 to 480. When comparing constraint-

based methods, SATMAP and TB-OLSQ2 findings suggest that TB-OLSQ2 outperforms SATMAP runtime. In a head-to-head comparison between constraint-based and heuristic methods, a great difference in runtime is observed. SABRE, representing heuristic methods, is on average about 80 times faster than the constraint-based TB-OLSQ2. When comparing heuristic methods, QMAP and TOQM are faster than Qiskit's methods. QMap is about 15 times faster than SABRE in tranpiling smaller circuits. It is also important to note that Qiskit's methods performs very similar in terms of runtime.

Table 6.4 shows that constraint-based methods in average adds less swap gates than heuristic methods. While TB-OLSQ2 performs better in constraint based methods, SABRE outperforms other heuristic methods. When comparing SABRE and TB-OLSQ2 only in one circuit named alu-bdd_288, TB-OLSQ2 adds two less swap gates and in one circuit named alu-v4_37 SABRE adds one less swap gate. Although QMap is way faster than SABRE in terms of transpilation time, It adds approximately 15 times more swap gates in smaller circuits.

Circuit Name	Cir. Size	sabre	noise	dense	trivial	toqm	satmap	tb-olsq2	qmap
graycode6_47	5	0	0	6	4	0	0	0	0
ex1_226	7	0	0	6	5	1	0	0	1
xor5_254	7	0	0	6	4	1	0	0	1
4gt11_84	18	0	0	4	4	3	0	0	3
ex-1_166	19	0	0	4	2	3	0	0	3
ham3_102	20	0	0	4	2	3	0	0	3
4mod5-v0_20	20	0	0	4	5	1	0	0	1
4mod5-v1_22	21	0	0	7	6	1	0	0	3
4gt11_83	23	0	1	7	8	2	0	0	4
4gt11_82	27	1	1	6	6	3	1	1	4
rd32-v0_66	34	0	0	7	5	3	0	0	3
4mod5-v0_19	35	0	0	5	10	2	0	0	1
mod5mils_65	35	0	0	4	5	3	0	0	0
rd32-v1_68	36	0	0	7	5	3	0	0	3
4mod5-v1_24	36	0	0	8	7	9	0	0	4
alu-v0_27	36	1	2	9	6	5	1	1	4
3_17_13	36	0	0	4	6	6	0	0	6
alu-v2_33	37	1	3	9	5	7	1	1	5
alu-v1_29	37	1	2	8	6	5	1	1	5
alu-v1_28	37	1	3	9	7	6	1	1	4
alu-v3_35	37	1	1	8	10	5	1	1	3
alu-v4_37	37	0	2	8	10	5	1	1	4
miller_11	50	0	0	4	6	9	0	0	9
decod24-v0_38	51	0	0	8	8	6	0	0	6
decod24-v2_43	52	0	0	9	10	6	0	0	6
alu-v3_34	52	1	4	10	5	6	1	1	6
4gt13_92	66	0	5	12	14	12	0	0	18
4gt13-v1_93	68	0	5	11	16	12	0	0	13
4mod5-v0_18	69	1	2	11	10	7	2	1	10
4mod5-v1_23	69	1	4	11	14	9	4	1	10
4mod5-bdd_287	70	1	5	10	10	6	2	1	10
qe_qft_4	71	0	2	0	6	2	3	0	3
4gt5_75	83	2	6	15	19	17	2	2	7
alu-bdd_288	84	4	6	15	14	15	7	2	13
decod24-v1_41	85	1	9	12	16	14	1	1	23
ising_model_10	480	0	2	15	7	7	0	0	30

Table 6.4: Number of Swap Gates Added to Each Circuit After Transpilation Process for Different Mapping and Routing Algorithms

6.2.2 Evaluation with Large Circuit Batch

Since constraint-based methods are not feasible for larger circuits, this evaluation with larger circuits only compares heuristic methods.

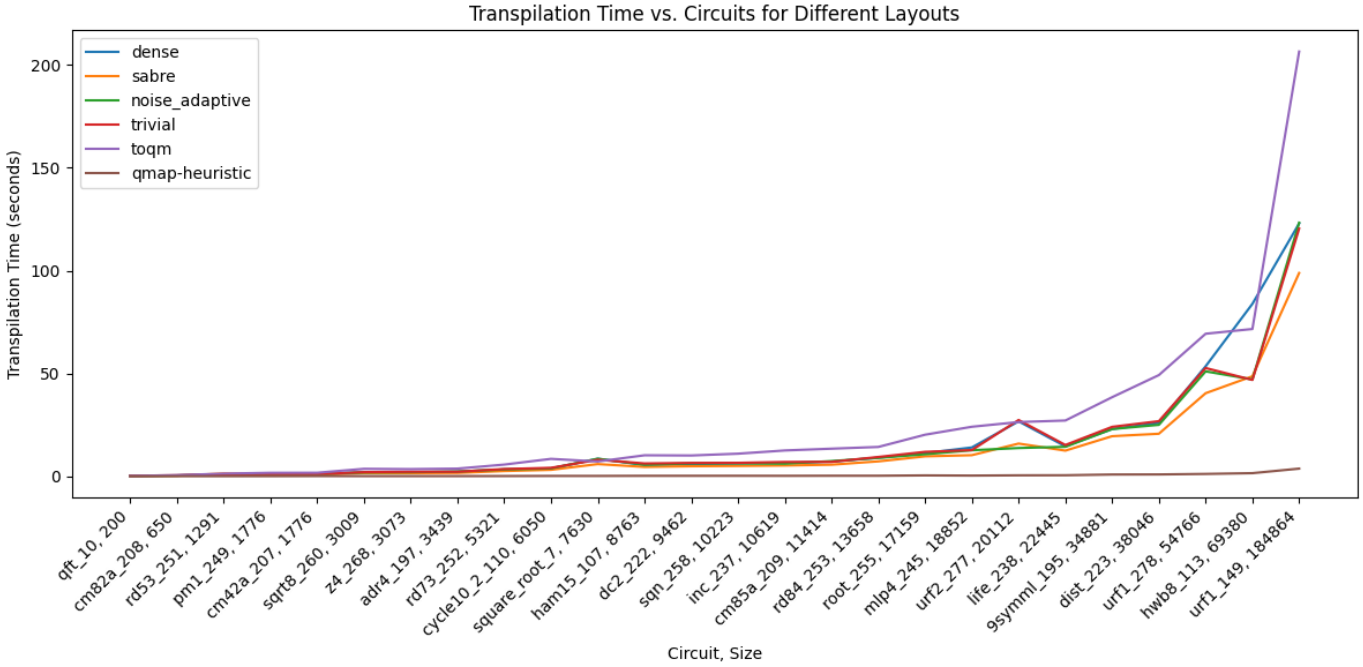


Figure 6.5: Runtime of Different Heuristic Mapping and Routing Algorithms vs Larger Quantum Circuits

Figure 6.5 shows runtime of different heuristic mapping and routing algorithms for larger Quantum circuits varying from 200 to 184864. While TOQM has slower runtime from the other heuristic methods, QMAP significantly outperforms and is 50 times faster than second fastest algorithm SABRE.

Table 6.5 shows the number of swap gates added to each circuit after transpilation process for different heuristic methods. SABRE adds the least amount of swap gates, just like it was in the smaller circuits. While QMAP is faster, it adds the most swap gates in all methods, approximately 1.5 times more than SABRE.

Circuit Name	Circuit Size	sabre	noise_adaptive	dense	trivial	toqm	qmap
qft_10	200	9	9	21	14	22.0	25
cm82a_208	650	26	30	47	64	44.0	76
rd53_251	1291	63	92	108	131	118.0	133
pm1_249	1776	75	139	111	175	125.0	235
cm42a_207	1776	76	141	131	164	125.0	235
sqrt8_260	3009	241	268	293	314	407.0	435
z4_268	3073	213	303	269	325	350.0	330
adr4_197	3439	268	350	324	324	378.0	557
rd73_252	5321	431	465	513	564	565.0	662
cycle10_2_110	6050	470	537	565	625	928.0	1055
square_root_7	7630	582	644	712	719	884.0	1106
ham15_107	8763	740	775	680	893	952.0	1395
dc2_222	9462	793	903	805	895	1011.0	1457
sqn_258	10223	768	923	805	1066	1100.0	1853
inc_237	10619	775	799	861	973	1142.0	1549
cm85a_209	11414	925	1085	907	1096	1250.0	1848
rd84_253	13658	1271	1246	1271	1521	1533.0	2074
root_255	17159	1424	1398	1482	1760	2221.0	2833
mlp4_245	18852	1655	1913	1887	2083	2379.0	2906
urf2_277	20112	2404	2225	2299	2593	2803.0	2936
life_238	22445	1992	2019	2108	2599	3072.0	3746
9symml_195	34881	2910	3500	3296	3798	3987.0	6164
dist_223	38046	3157	3274	3280	4055	5265.0	6425
urf1_278	54766	6020	6113	6322	6697	7439.0	7643
hwb8_113	69380	6664	6568	5888	7179	6498.0	11342
urf1_149	184864	17966	18346	18863	20138	20408.0	21374

Table 6.5: Number of Swap Gates Added to Each Circuit After Transpilation Process for Different Mapping and Routing Algorithms

6.3 Evaluation of Transpilation with a Sub-Architecture

In this section, the effectiveness of employing sub-architectures in the transpilation process is assessed. The evaluation focuses on two primary aspects: the runtime of the transpilation process and the impact on the quality of quantum circuits, particularly in terms of swap gates added.

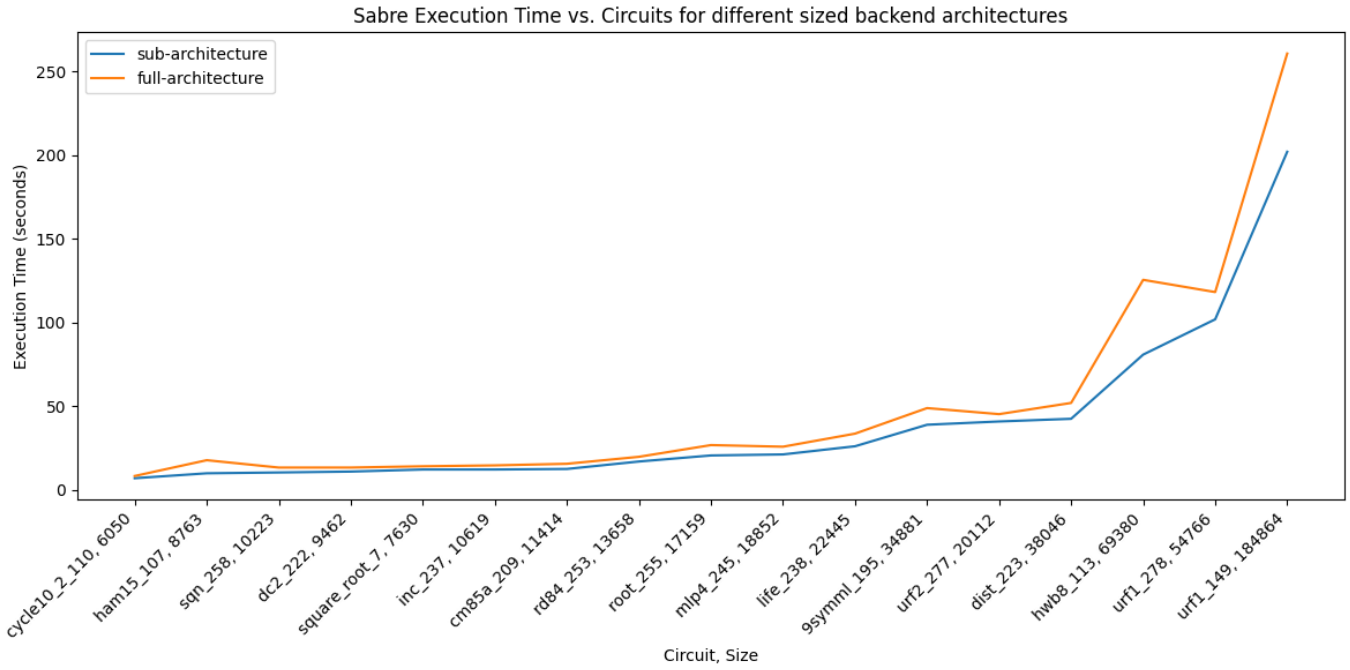


Figure 6.6: Runtime of SABRE vs Quantum Circuits for Full-architecture of IBM-Kolkata and Sub-architecture of IBM-Kolkata

Figure 6.6 present the runtime comparisons between full and sub-architectures of IBM Kolkata when using SABRE as the layout method for transpilation. The size of quantum circuits used ranges from 6050 to 184864. Size here indicates number of gates in the circuit. Using sub-architecture is 1.25x faster than using the full architecture on average.

Table 6.6 detail the number of swap gates added post-transpilation. The results indicate that sub-architectures can almost maintain the quality of circuits with only %3 increase in added swap gates on average.

Name	Circuit Size	Sub-arch	Full-arch
cycle10_2_110	6050	1036	947
square_root_7	7630	1438	1479
ham15_107	8763	1542	1530
dc2_222	9462	1689	1679
sqn_258	10223	1864	1912
inc_237	10619	1753	1754
cm85a_209	11414	2208	1874
rd84_253	13658	2622	2588
root_255	17159	3564	3119
mlp4_245	18852	3288	3331
urf2_277	20112	5382	4894
life_238	22445	4391	4008
9symml_195	34881	6903	6896
dist_223	38046	7913	6980
urf1_278	54766	12680	12761
hwb8_113	69380	12561	12538
urf1_149	184864	33547	33306

Table 6.6: Number of Swap Gates Added to Each Circuit After Sabre for Full-architecture of IBM-Kolkata and Sub-architecture of IBM-Kolkata

Figure 6.7 provides a comparison of the runtime between the full and sub-architectures of IBM Kolkata when employing the SATMAP method for transpilation. The circuits used in this experiment varied in size from 7 to 84 gates. Remarkably, the use of sub-architectures leads to a tenfold average improvement in runtime. This significant reduction can be attributed to the diminished solution space for the SAT solver. By focusing on a subset of the full architecture, the complexity of the optimization problem is considerably reduced, leading to faster transpilation times.

Table 6.7 detail the number of swap gates added post-transpilation. The results shows that sub-architectures either maintain or enhance circuit quality compared to the full architecture. It is important to remind here that since satmap implements a local relaxcation for maxsat solver, It does not ensure global optimality across the entire circuit.

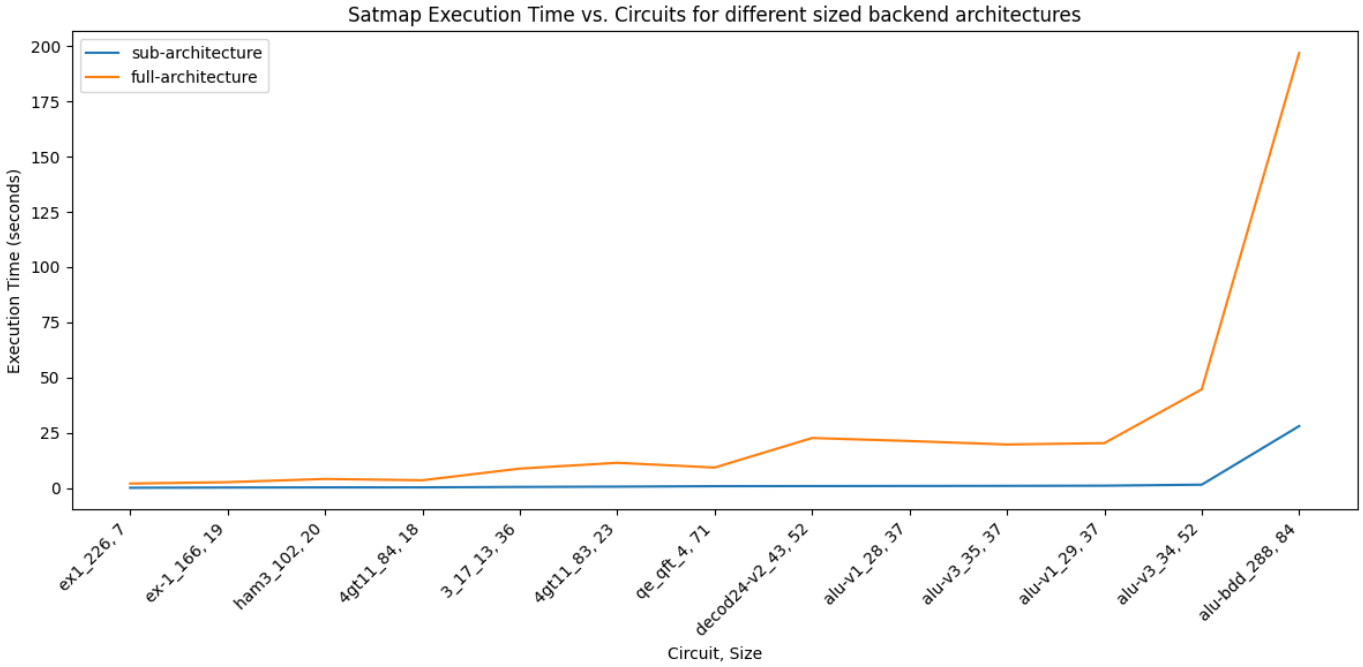


Figure 6.7: Runtime of Satmap vs Quantum Circuits for Full-architecture of IBM-Kolkata and Sub-architecture of IBM-Kolkata

Name	Full-arch	Sub-arch
qe_qft_4	3	3
alu-bdd_288	23	21
ham3_102	3	3
alu-v3_35	7	7
3_17_13	6	6
alu-v1_29	7	7
qe_qft_4	3	3
3_17_13	6	6
4gt11_83	5	5
4gt11_84	3	3
ex1_226	2	2
ex-1_166	3	3

Table 6.7: Number of Swap Gates Added to Each Circuit After Satmap for Full-architecture of IBM-Kolkata and Sub-architecture of IBM-Kolkata

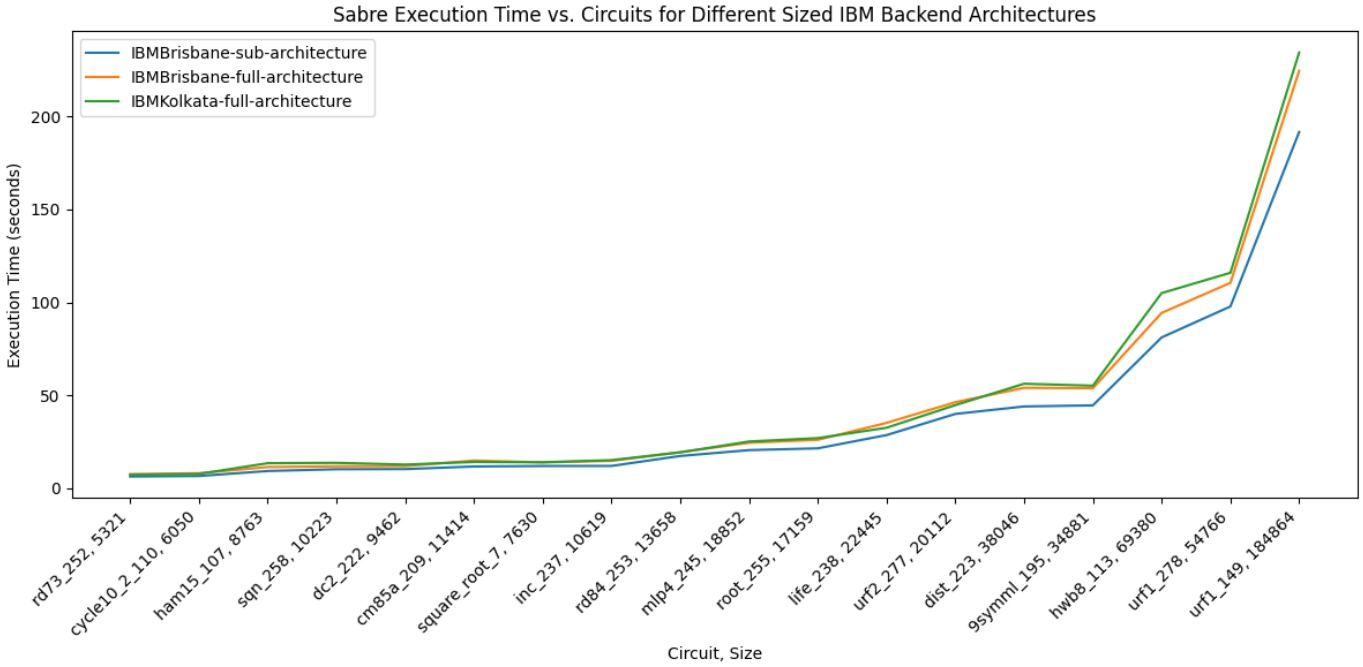


Figure 6.8: Transpilation Runtime vs Quantum Circuits for Different Sized IBM Backend Architectures

Figure 6.8 provides a comparison of the runtime between the full-architectures of IBM Kolkata and IBM Brisbane and a 27-qubit subarchitecture of IBM Brisbane when employing the SABRE method for transpilation. The size of quantum circuits used ranges from 5321 to 184864. Size here indicates number of gates in the circuit. Using a 27-qubit subarchitecture of IBM Brisbane if 1.16x faster on average than using full architecture of IBM Brisbane and 1.2x faster than using IBM Kolkata.

Table 6.8 detail the number of swap gates added post-transpilation. The results shows that sub-architectures can almost maintain the quality of circuits.

Name	Num Qubits	Circuit Depth	Circuit Size	Sub-Arch*	IBM Brisbane	IBM Kolkata
square_root_7	15	3847	7630	1516	1486	1474
dc2_222	15	5242	9462	1656	1650	1659
cycle10_2_110	12	3386	6050	965	947	959
urf1_278	9	30955	54766	12599	12425	12602
cm85a_209	14	6374	11414	1964	1929	1867
urf1_149	9	99585	184864	33711	33662	34693
sqn_258	10	5458	10223	1848	1829	1933
mlp4_245	16	10328	18852	3317	3296	3288
ham15_107	15	4819	8763	1552	1553	1565
hwb8_113	9	38717	69380	12845	12759	12676
rd84_253	12	7261	13658	2627	2615	2608
urf2_277	8	11390	20112	4865	5009	4979
root_255	13	8835	17159	3174	3161	3126
rd73_252	10	2867	5321	994	1004	1008
life_238	11	12511	22445	4047	4034	4012
9symml_195	11	19235	34881	6577	6608	6988
dist_223	13	19694	38046	7020	7058	6999
inc_237	16	5863	10619	1761	1795	1787

Table 6.8: Number of Swap Gates Added to Each Circuit After Transpilation Process for Different Sized IBM Backend Architectures. (*Sub-Arch refers to a sub-architecture of IBM Brisbane Backend with 27 qubits)

7 Related Work

7.1 Qubit Mapping and Routing Surveys

[20] reviews two existing transpilation methods [19] [22], and suggests how they can be optimized using heuristic as well as exact methods.

[6] presents a comprahensive survey on Quantum Circuit Compilation (QCC) specifically focusing on artificial intelligence (AI)-based and heuristic-based methods. They group them based on underlying techniques that they implement, such as AI algorithms including genetic algorithms, genetic programming, ant colony optimization and AI planning, and heuristics methods employing greedy algorithms, satisfiability problem solvers, dynamic, and graph optimization techniques. This work discusses performance of each QCC technique and evaluate its potential limitations but it does not provide any experimantal results.

[16] uses QUEKO benchmarks to evaluate the optimality of current layout synthesis tools, including Cirq from Google, Qiskit from IBM, tiket from Cambridge Quantum Computing. It demonstrate large optimality gaps: 1.5-12x on average on a smaller device and 5-45x on average on a larger device.

7.2 Transpilation with Subarchitectures

The research in [19] focuses on using Boolean satisfiability solvers for the qubit mapping problem in quantum computing, proposing an optimization by considering a quantum computing backend's subarchitecture. This involves selecting the best-performing subarchitecture from all possible ones with a qubit count matching the circuit. While this method is computationally intensive, the reduced search space for the SAT solver notably speeds up the solving process.

[1] demonstrates that optimal results in mapping can still be achieved by significantly limiting the search space. It follows a similar subarchitecture selection approach as [19].

[13] counters the notion that minimally restricting the search space ensures optimality, as suggested in [19] and [1]. It delves into the complexity of identifying minimal-size subarchitectures for optimal mapping and introduces relaxed criteria for practical circuits. The study offers methods for finding near-optimal subarchitectures to improve

quantum circuit mapping efficiency.

This work, while not guaranteeing optimality, presents a fidelity-based subarchitecture selection process, differing from the previously mentioned studies.

8 Summary & Conclusion

8.1 Summary of Work

This thesis presented a comprehensive study of quantum computing transpilation, focusing on the optimization of quantum backend usage and the evaluation of state-of-art qubit mapping and routing algorithms. The research explored the efficiency and scalability of Qiskit's transpiler, investigated the performance of different quantum mapping and routing algorithms, and assessed the effectiveness of employing sub-architectures in the transpilation process. The experimental framework utilized IBM quantum backends and encompassed a range of quantum circuits with varying sizes and complexities.

8.2 Conclusion

The key findings from this research are summarized as follows:

Evaluation of Qiskit's Transpiler: The study revealed that higher optimization levels in Qiskit's transpiler lead to an increased transpilation runtime but result in circuits with fewer swap gates. The scalability analysis showed a significant computational bottleneck with larger circuits, especially at the highest optimization level.

Comparison of Quantum Mapping and Routing Algorithms: A comparative analysis of various algorithms demonstrated that heuristic approaches, particularly QMap and SABRE, are significantly faster than constraint-based methods like SATMap and TB-OLSQ2. However, SABRE consistently added the fewest swap gates compared to other heuristic methods. Constraint-based methods generally produced circuits with fewer swap gates but were less scalable for larger circuits.

Transpilation with Sub-Architectures: The use of sub-architectures significantly improved the runtime of the transpilation process, both for the SABRE layout algorithm and the SATMap method. While there was a slight increase in the number of swap gates added, the overall quality of the circuits was largely maintained.

The findings highlight the trade-offs between transpilation time, circuit quality, and scalability in quantum computing. They underscore the importance of selecting appropriate backends and optimization strategies for efficient quantum computation.

This research contributes to the ongoing efforts in optimizing quantum algorithms and hardware, paving the way for more practical and efficient quantum computing applications. In conclusion, this thesis provides valuable insights into the challenges and opportunities in quantum circuit transpilation. The research underscores the importance of backend selection, optimization levels, and algorithm choice in the context of quantum computing. By exploring various dimensions of quantum transpilation, the study contributes to the advancement of quantum computing, offering guidelines and benchmarks for future research and practical applications in this rapidly evolving field.

9 Future Work

The results in Figure 4.1 showed that other passes in Qiskit's transpilation process such as Unitary Synthesis and Consolidate Blocks takes as much time as Sabre Layout. Future research could explore various strategies to enhance the efficiency of these methods.

In addition to heuristic and constraint-based qubit mapping and routing algorithms, analyzing machine learning techniques to predict optimal transpilation strategies based on circuit characteristics and backend specifications can be done.

The thesis proposes selecting a subarchitecture with higher fidelity for quantum circuit transpilation, but a key area for future research involves practical testing. This would involve executing the quantum circuits on real quantum hardware to verify if the higher fidelity subarchitectures lead to improved outcomes.

Abbreviations

NISQ Noisy Intermediate-Scale Quantum

QEC Quantum Error Correction

qubits Quantum Bits

QASM Quantum Assembly Language

CNOT Controlled NOT

SAT Boolean satisfiability problem

MAXSAT Maximum Satisfiability

SMT Satisfiability Modulo Theories

BIP Binary Integer Program

SABRE SWAP-based Bidirectional heuristic search algorithm

TOQM Time-Optimal Qubit Mapping

OLSQ Optimal Layout Synthesis using Quantum Computing

TB-OLSQ Transition-Based OLSQ

QCC Quantum Circuit Compilation

List of Figures

List of Tables

Bibliography

[1] Lukas Burgholzer, Sarah Schneider, and Robert Wille. "Limiting the Search Space in Optimal Quantum Circuit Mapping." In: *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2022, pp. 466–471. DOI: 10.1109/ASP-DAC52403.2022.9712555.

[2] Alexander Cowtan et al. "On the Qubit Routing Problem." en. In: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. DOI: 10.4230/LIPICS.TQC.2019.5. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.TQC.2019.5.

[3] Leonardo De Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver." In: *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. TACAS'08/ETAPS'08. Budapest, Hungary: Springer-Verlag, 2008, pp. 337–340. ISBN: 3540787992.

[4] IBM. *How Do Quantum Computers Work?* 2023. URL: https://www.ibm.com/topics/quantum-computing#How+do+quantum+computers+work%3F%09%09%09%09%09%09%09.

[5] IBM. *What is Quantum Computing?* 2023. URL: https://www.ibm.com/topics/quantum-computing.

[6] Janusz Kusyk, Samah M. Saeed, and Muharrem Umit Uyar. "Survey on Quantum Circuit Compilation for Noisy Intermediate-Scale Quantum Computers: Artificial Intelligence to Heuristics." In: *IEEE Transactions on Quantum Engineering* 2 (2021), pp. 1–16. DOI: 10.1109/TQE.2021.3068355.

[7] Gushu Li, Yufei Ding, and Yuan Xie. "Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices." 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. 1001–1014. ISBN: 9781450362405. DOI: 10.1145/3297858.3304023. URL: https://doi.org/10.1145/3297858.3304023.

[8] Wan-Hsuan Lin et al. "Scalable Optimal Layout Synthesis for NISQ Quantum Processors." In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 2023, pp. 1–6. DOI: 10.1109/DAC56929.2023.10247760.

[9] Abtin Molavi et al. "Qubit Mapping and Routing via MaxSAT." In: *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2022, pp. 1078–1091. DOI: 10.1109/MICRO56248.2022.00077.

[10] Prakash Murali et al. "Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale 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. 1015–1029. ISBN: 9781450362405. DOI: 10.1145/3297858.3304075. URL: https://doi.org/10.1145/3297858.3304075.

[11] Giacomo Nannicini et al. "Optimal Qubit Assignment and Routing via Integer Programming." In: *ACM Transactions on Quantum Computing* 4.1 (Oct. 2022). DOI: 10.1145/3544563. URL: https://doi.org/10.1145/3544563.

[12] Paul D. Nation and Matthew Treinish. "Suppressing Quantum Circuit Errors Due to System Variability." In: *PRX Quantum* 4 (1 Mar. 2023), p. 010327. DOI: 10.1103/PRXQuantum.4.010327. URL: https://link.aps.org/doi/10.1103/PRXQuantum.4.010327.

[13] Tom Peham, Lukas Burgholzer, and Robert Wille. "On Optimal Subarchitectures for Quantum Circuit Mapping." In: *ACM Transactions on Quantum Computing* 4.4 (July 2023), pp. 1–20. ISSN: 2643-6817. DOI: 10.1145/3593594. URL: http://dx.doi.org/10.1145/3593594.

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

[15] Eleanor Rieffel and Wolfgang Polak. "An Introduction to Quantum Computing for Non-Physicists." In: *ACM Comput. Surv.* 32.3 (Sept. 2000), pp. 300–335. ISSN: 0360-0300. DOI: 10.1145/367701.367709. URL: https://doi.org/10.1145/367701.367709.

[16] B. Tan and J. Cong. "Optimality Study of Existing Quantum Computing Layout Synthesis Tools." In: *IEEE Transactions on Computers* 70.09 (Sept. 2021), pp. 1363–1373. ISSN: 1557-9956. DOI: 10.1109/TC.2020.3009140.

[17] Bochen Tan and Jason Cong. "Optimal Layout Synthesis for Quantum Computing." In: *Proceedings of the 39th International Conference on Computer-Aided Design*. ICCAD '20. Virtual Event, USA: Association for Computing Machinery, 2020. ISBN: 9781450380263. DOI: 10.1145/3400302.3415620. URL: https://doi.org/10.1145/3400302.3415620.

[18] Swamit S. Tannu and Moinuddin 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.

[19] Robert Wille, Lukas Burgholzer, and Alwin Zulehner. "Mapping Quantum Circuits to IBM QX Architectures Using the Minimal Number of SWAP and H Operations." In: *Proceedings of the 56th Annual Design Automation Conference 2019*. DAC '19. ACM, June 2019. DOI: 10.1145/3316781.3317859. URL: http://dx.doi.org/10.1145/3316781.3317859.

[20] Robert Wille, Stefan Hillmich, and Lukas Burgholzer. "Efficient and Correct Compilation of Quantum Circuits." In: *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2020, pp. 1–5. DOI: 10.1109/ISCAS45731.2020.9180791.

[21] Chi Zhang et al. "Time-Optimal Qubit Mapping." In: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '21. Virtual, USA: Association for Computing Machinery, 2021, pp. 360–374. ISBN: 9781450383172. DOI: 10.1145/3445814.3446706. URL: https://doi.org/10.1145/3445814.3446706.

[22] Alwin Zulehner, Alexandru Paler, and Robert Wille. *An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures*. 2018. arXiv: 1712.04722 [quant-ph].