Crosstalk-aware NISQ Multi-programming

Yasuhiro Ohkura

A thesis presented for the degree of
Bachelor of Policy Management

Faculty of Policy Management
Keio University

Abstract

Current quantum processors are known as Noisy Intermediate-Scale Quantum (NISQ) devices and are greatly susceptible to noise. In particular, gate errors that accumulate every operation and the qubit lifetime are known to be major sources of noise, that limit the scale of reliable quantum computation. On the other hand, the state of the art quantum processor consists of up to tens of qubits, and some of them are available via cloud services. As a result, the number of underutilized resources increases due to the excess number of qubits over the size of the problem that can be solved reliably, which limits the throughput of the NISQ computation. Parallel execution of small-scale circuits may improve the operating efficiency of NISQ processors, but still challenging because the impact of errors on individual tasks is non-trivial. In particular, the crosstalk that is known as the non-local error in the quantum chip can cause unwanted interference between independent circuits in a simultaneous execution set, and it may lead to limit the performance of parallel execution. In this paper, we propose a novel compilation method to mitigate the impact of crosstalk during concurrent execution of multiple circuits and to realize highly reliable quantum computation. We use IBM Q System 27-qubit processor to characterize the crosstalk noise and to evaluate the performance of our proposal. This method provides more efficient and highly reliable quantum computation by effectively allocating multiple quantum circuits to the processor, taking into account unwanted remote interference caused by crosstalk. Our work improves the throughput of the NISQ processor and achieves high-speed processing of small tasks. This would be attractive not only to quantum providers but also to users around the world who want to run the small-scale NISQ algorithms that have recently attracted great focus and are being enthusiastically investigated.

Contents

1 Introduction **3**

2 Background **5**
2.1 Principles of Quantum Computation 5
2.2 Noisy Intermediate-Scale Quantum Computing 5
2.2.1 Errors in NISQ 7
2.3 Quantum Circuit 7
2.3.1 Compilation task 8
2.4 Cloud Quantum Computing Services 9
2.5 Crosstalk in NISQ processor 9
2.5.1 The cost of noise characterization 9
2.5.2 Noise mitigation approach 9
2.6 Quantum Multi-programming 10

3 Error Detection Methodology **11**
3.1 Crosstalk Noise 11
3.1.1 Experimental setup 11
3.1.2 Characterization Crosstalk Noise 12
3.1.3 Reducing detection overhead strategy 15
3.2 Measurement Waiting Duration 17
3.2.1 Experimental setup 17
3.2.2 Impact of measurement waiting duration 17

4 Proposed Algorithms **20**
4.1 Qubit allocation policy 20
4.1.1 High cost circuit first allocation 20
4.1.2 Crosstalk aware qubit allocation 21
4.2 Scheduling multi-programming instruction policy 22

5 Experiments and Evaluation **23**
5.1 Setup 23
5.2 Crosstalk adaptive layout 24
5.2.1 Results 24
5.3 Modifying Measurement Waiting Duration 29
5.3.1 Evaluation with benchmark circuits 29

6 Conclusion and Remarks **34**

Chapter 1

Introduction

Quantum computing aims to effective information processing on specific problems by utilizing the properties of quantum mechanics, such as superposition and entanglement in computer technology. It is, in particular, expected to be applied in the fields of chemistry, finance, and machine learning. On the contrary, the current quantum processor what is called Noisy Intermediate-Scale Quantum (NISQ) [1], is not immune to the noise which causes a high error rate and greatly affects the reliability of the computation.

With the advent of the cloud quantum computing system [2], [3], quantum computing has become familiar to researchers and developers around the world. The more number of users and tasks from various demands and backgrounds has increased, the more it is important that the throughput of NISQ processor. To operate this efficiently, executing multiple quantum tasks concurrently can be one solution however this method is not trivial and involves essential challenges [4], [5].

It is difficult to explain whole errors of computation on NISQ processor by using information from standard error characterization techniques such as quantum tomography and randomized benchmarking [6], [7]. To maximize the utilization and performance of NISQ processor, we should take into account not only the standard one also context-dependent errors [8].

In the case of superconducting qubit systems, crosstalk has the biggest effect on the gate errors [9], [10]. When multiple quantum circuits are executed in parallel, as resource usage of the processor increases, unwanted interference may occur due to crosstalk noise between independent quantum circuits, which may affect the calculation results [4], [5]. In this research, we propose a novel compilation method for mitigating crosstalk noise that assumes parallel execution of multiple quantum circuits.

This thesis includes the following contributions. First, we visualize how the impact of crosstalk distributed in the real device potentially limits the performance of computation and the efficient way of the noise characterization method in practice. Second, we propose a novel qubit allocation method for the quantum multi-programming case that is considering crosstalk characterization in the hardware to improve output fidelity. Third, we propose a scheduling method for the simultaneous execution of multi circuits.

Our work consists of the following. At first, we have attempted to clarify the problem and motivation of this study referring to previous works (Ch.2). As the main contribution of this thesis, first, we characterize and analyze the crosstalk noise and coherent data that are potentially limiting the performance of multi-programming (Ch.3). Second, we propose a novel qubit allocation method considering crosstalk characterization in the hardware and scheduling method cares about the difference of duration of each independent circuit executed concurrently to improve output fidelity, and show the performance of our proposal (Ch.4, 5).

Chapter 2

Background

2.1 Principles of Quantum Computation

The unit of the quantum computer is called a quantum bit, i.e. ***qubit*** described as state vector $|0\rangle$ and $|1\rangle$. Qubit has the capability of taking superposition state as $\alpha|0\rangle + \beta|1\rangle$ against the classical counter part only have 0 and 1 state. Here α, β are complex numbers that satisfy the $|\alpha|^2 + |\beta|^2 = 1$. In quantum computing, we can utilize special correlations between qubits called entanglement that particular to quantum physics. In general, quantum algorithms are executed as follows. 1). Initialize a qubit state as superposition. 2). Apply quantum operations to qubits. In the quantum circuit model (Sec. 2.3), those operations are implemented combination of quantum gates Table 2.1 in time steps. 3). Finally, by applying measurement operation, we can extract the final state as a classical probability distribution of bit strings.

2.2 Noisy Intermediate-Scale Quantum Computing

The performance of current leading quantum architectures called Noisy Intermediate-Scale Quantum (NISQ) is limited due to the noise [1]. The providers are continuously building better NISQ processors both in size and error tolerance[12], [13]. Nevertheless, qubits are affected by noise that causes serious errors in the calculation due to the reasons such as gate operation, measurement operation, and environmental interference. With Quantum Error Correction (QEC) technique, the promised quantum algorithms including Shor's algorithm [14], Grover's algorithm [15] provide the

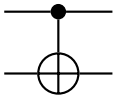
Quantum Gate	Circuit diagram	Matrix representation	Truth table	
			Input	Output
Identity gate	$\boxed{Id}$	$Id = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\|0\rangle$ $\|1\rangle$	$\|0\rangle$ $\|1\rangle$
X gate	$\boxed{X}$	$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$\|0\rangle$ $\|1\rangle$	$\|1\rangle$ $\|0\rangle$
Y gate	$\boxed{Y}$	$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$\|0\rangle$ $\|1\rangle$	$i\|1\rangle$ $-i\|0\rangle$
Z gate	$\boxed{Z}$	$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$\|0\rangle$ $\|1\rangle$	$\|0\rangle$ $-\|1\rangle$
S gate	$\boxed{S}$	$S = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{pmatrix}$	$\|0\rangle$ $\|1\rangle$	$\|0\rangle$ $e^{i\frac{\pi}{2}}\|1\rangle$
T gate	$\boxed{T}$	$S = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}$	$\|0\rangle$ $\|1\rangle$	$\|0\rangle$ $e^{i\frac{\pi}{4}}\|1\rangle$
H gate	$\boxed{H}$	$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	$\|0\rangle$ $\|1\rangle$	$\frac{\|0\rangle+\|1\rangle}{\sqrt{2}}$ $\frac{\|0\rangle-\|1\rangle}{\sqrt{2}}$
Control-NOT gate		$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	$\|0\rangle\|0\rangle$ $\|0\rangle\|1\rangle$ $\|1\rangle\|0\rangle$ $\|1\rangle\|1\rangle$	$\|0\rangle\|0\rangle$ $\|0\rangle\|1\rangle$ $\|1\rangle\|1\rangle$ $\|1\rangle\|0\rangle$

Table 2.1: **Quantum Gate examples [11]**.

quantum advantage against classical counterparts. However, QEC requires physical qubits overhead [16], [17]. It is quite hard to succeed in the NISQ era. Instead, to achieve the quantum advantage earlier, *near-term Quantum Computation* [18] aims to develop the software and algorithms for the quantum hardware available in the next few years without QEC including Variational Quantum Algorithms [19], [20], [21], Variational Quantum Eigensolver [22], [23], Quantum Approximate Optimization Algorithm [24], Approximated post-NISQ algorithm [25]. In this thesis, we focus on software development for efficient usage of the NISQ processor.

2.2.1 Errors in NISQ

In physically, the NISQ processors are implemented as an open quantum system. The qubits interact with their environments, and this noise arises as a quantum error that makes a quantum state fragile [11]. Though there are several reasons to cause the errors, we introduce operational errors and the decoherence caused by qubit lifetime here in part. Due to the limited precision of the quantum operation implemented in a real architecture, every time we perform quantum gates or measurements to the qubits, the quantum state becomes different from ideal with a certain probability, and errors accumulate in time steps. We call these gate errors and readout errors. In the IBM Q System, for instance, two-qubit ($CNOT$ gate) error rates are around 10x bigger than one-qubit gate error rates. Another source of decoherence comes from qubit lifetime. Qubit coherence time represented as T_1, T_2 and its related metric T_2* [26]. T_1 explains thermal relaxation time or amplitude damping that is decaying of quantum states from the excited state (higher energy state, $|1\rangle$) to the ground state (lower energy, $|0\rangle$) along with energy loss. T_2 and T_2* explain phase damping that is decaying of stable phase information.

2.3 Quantum Circuit

Quantum circuit is the model of quantum computation, that is composed of the quantum register(s) and classical register(s) described as line(s), series of the quantum gate(s) as box(es), and measurement operation described as meter connected to classical register(s) Fig. 2.1. Every quantum gate is defined as the unitary operation that means the quantum operation is reversible. On the contrary, the measurement of qubit(s) is a destructive and irreversible operation. It collapses the quantum state and converts it to classical (on/off) form. In this process, though a lot of information represented in quantum space is destroyed, we can observe what it was probabilistically. To understand the final product of the series of quantum gate operations by

measurement, we conduct thousands of trials and produce a result as a probability distribution of bit-strings in a real system.

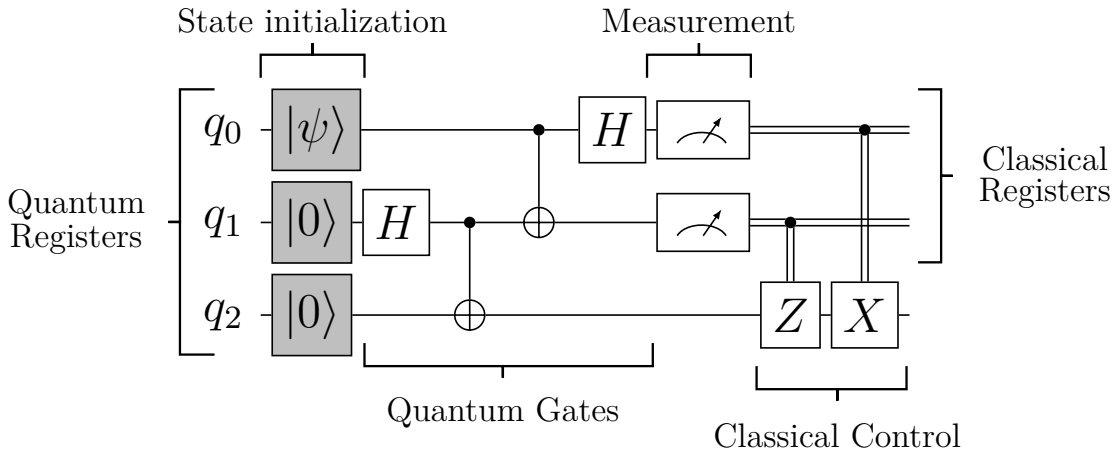


Figure 2.1: **Example of Quantum Circuit: Quantum Teleportatin [27].**

2.3.1 Compilation task

To realize logical quantum computation described as a quantum circuit on the real-world physical system, it should be compiled to a proper form that satisfying logical and physical constraints. It is known that the quantum circuit consists of only a few elements include state preparation, Hadamard gates, phase gates, $CNOT$ gates, Pauli gates, measurement, and classically conditioned control gates by *Gottesman-Knill theorem* [28]. In the case of current leading quantum architectures, its own universal quantum gate set is implemented to realize arbitrary quantum circuit operations corresponding to the theorem. The compiler translates the program circuit written by the user to this gate set. This is called logic synthesis [29]. To satisfy the physical constraints of the quantum processor, program qubits are allocated to the physical system and gate operations are routed and scheduled considering the hardware topology. In the case of the NISQ processor, because of the noise and fragile quantum states, optimization of this circuit compilation process has a serious impact on the reliability of the output. The layout of the program qubits to hardware is related to the reduction of SWAP operation in the routing process. SWAP operation is implemented three $CNOT$ gates and this 2-qubit gates are major source of noise in the current system showed in Sec. 2.2.1. Previous works showed practical optimization techniques for allocation [30], [31], [29], routing [32], [33] for the NISQ system.

2.4 Cloud Quantum Computing Services

With the advent of cloud quantum computing services [34], [35], [36], [2], [37], [38], [39] , so many researchers and developers from a variety of domains are becoming quantum users. Cloud quantum computing provides great opportunities for conducting basic experiments to developing applications include quantum simulation, quantum machine learning, solving the optimization problem [40]. With the rapid increase in users, urgent accesses for limited cloud quantum resources and a number of queued jobs are becoming serious issues.

2.5 Crosstalk in NISQ processor

Crosstalk error is known as a significant source of noise in the quantum processor. This error can be explained from several aspects, but simply the unwanted interaction between coupled qubits in the processor. It is known there is a trade-off between the strength of qubits interaction and magnitude of unwanted crosstalk noise [41], [42]. And one type of crosstalk is caused by simultaneous operations between specific couples of qubits. In this thesis, we focused on the unwanted interaction due to the 2-qubits ($CNOT$ gate) operation. These types of crosstalk are known to occur in the current quantum architectures include the superconducting systems and trapped ion [43], [44].

2.5.1 The cost of noise characterization

The complexity of noise characterization often expands exponential scaling with the system size. Recently, several works show efficient ways to detect crosstalk noise [45]. Even so, it takes several hours for characterization, and it is an impractical recent situation, that is NISQ and cloud quantum computing era. Rather than that, we adapted a simpler protocol, simultaneous randomized benchmarking [41] and limited detection range that is introduced in Sec 3.1 to reduce characterization cost.

2.5.2 Noise mitigation approach

The tuning and mitigation of crosstalk directly become big challenges as developing larger processors [42], [8], [45]. There are several software approaches to reduce crosstalk error introduced in previous work. In the case of tunable quantum processors include Google's architecture[46], tune qubit frequencies or control specific couplers to disable and shutdown the leakage errors [9], [47]. On the contrary, for

fixed qubit systems include IBM Q System, by using optimized circuit scheduler to avoid concurrent execution of correlated qubits in the processor [48]. In this thesis, we focused on this fixed qubit system, and provide the solution by novel layout method in the circuit compilation process.

2.6 Quantum Multi-programming

The quantum multi-programming is the method to improve the throughput and utilization of the NISQ processor by executing multiple quantum circuits simultaneously, instead of keeping the unemployed qubits resource idle. In the previous work, several challenges were discussed as follows [4], [5]: 1). Fair hardware resource allocation for every individual task. The difficulty of this issue comes from the variations of characteristics of each physical qubits in the processor includes operational error rates and qubit lifetimes [49]. To solve this, the compiler needs to take those error information into account to optimize the circuit. 2). Avoidance of unwanted interference between the individual quantum circuit. Improving utilization of the processor by executing multiple programs concurrently can increase the unwanted interference between independent quantum circuits. To withdraw serious destructive interference, one option is to monitor and compare the performance of parallel execution and to feedback it to the next execution phase, single or parallel [4]. Rather than that, in this thesis, we directly focused on the crosstalk characterization in the processor and optimize qubit allocation itself. 3). Optimization of the operational timing of each circuit to minimize the unnecessary decoherence effect. In the case of multi-programming with several lengths of quantum circuits, the shorter circuits suffer wait duration until the longer circuit's operation ends, and it may cause the decaying of a quantum state prepared by shorter circuits and decline the output reliability. In this study, we focus on issues 2 and 3 in particular.

Chapter 3

Error Detection Methodology

3.1 Crosstalk Noise

3.1.1 Experimental setup

Error information of IBMQ systems from daily calibration is provided in IBM Quantum Experience [2]. We can utilize those for software or applications including noise-aware compiler [32]. However we need additional noise characterization to mitigate the crosstalk effect [48].

As we mentioned in Sec.2.5, one type of crosstalk occurs when we execute specific pairs of hardware qubits due to the strength of the coupling interaction. In this paper, we defined this effect as the ratio between a gate error rates single execution and conditional error rate more than two gates executed simultaneously. Namely, the crosstalk effect $X_{(g_i,g_j)}$ from g_j to the gate g_i defined as follows:

$$X_{(g_i,g_j)} = \frac{\epsilon_{(g_i|g_j)}}{\epsilon_{g_i}} \tag{3.1}$$

To detect crosstalk hardware, we use Randomized Benchmarking (RB), the standard method to characterize hardware gate error rates. There are several variations of this method depending on the situation and usage [6], [41], [50], [51], [10], [52].

First, we characterize gate error rates ϵ_{g_i} by the standard RB procedure. Then, to detect conditional error rates $\epsilon_{(g_i|g_j)}$, we use Simultaneous Randomized Benchmarking (SimRB) [41].

When we investigate the crosstalk error between gates g_i and g_j, we first calculate the respective error rates $E(g_i)$ and $E(g_j)$ using ordinary RB. Next, we calculate the conditional error rates $E(g_i|g_j)$ and $E(g_j|g_i)$ when two gates are executed simulta-

neously using SimRB. When g_i is affected by crosstalk by g_j, $E(g_i|g_j)$ is larger than $E(g_i)$.

3.1.2 Characterization Crosstalk Noise

Here we show the crosstalk magnitude and distribution on IBM Q System 27-qubit processor (IBM Q Toronto, IBM Q Paris) characterized by using SimRB introduced in Sec 3.1. Each cell represents the combination of conditioned error rates between the vertical and horizontal axis. Here we show the ratio of error rates of conditional cases and independent cases as the size of colored boxes. We can see that there are two types of crosstalk distribution. The one is a short-range and strong error, and the other is a long-range and weak error. The first one that the combinations placed in a few hops are a relatively higher crosstalk error, and also crosstalk appears symmetrically in the processor, for example, crosstalk between qubits $(7, 10)$ and $(12, 15)$ interact each other. On the other hand, crosstalk errors on $(19, 22)$ distribute long-range but relatively weak that also shown in Fig. 3.2d.

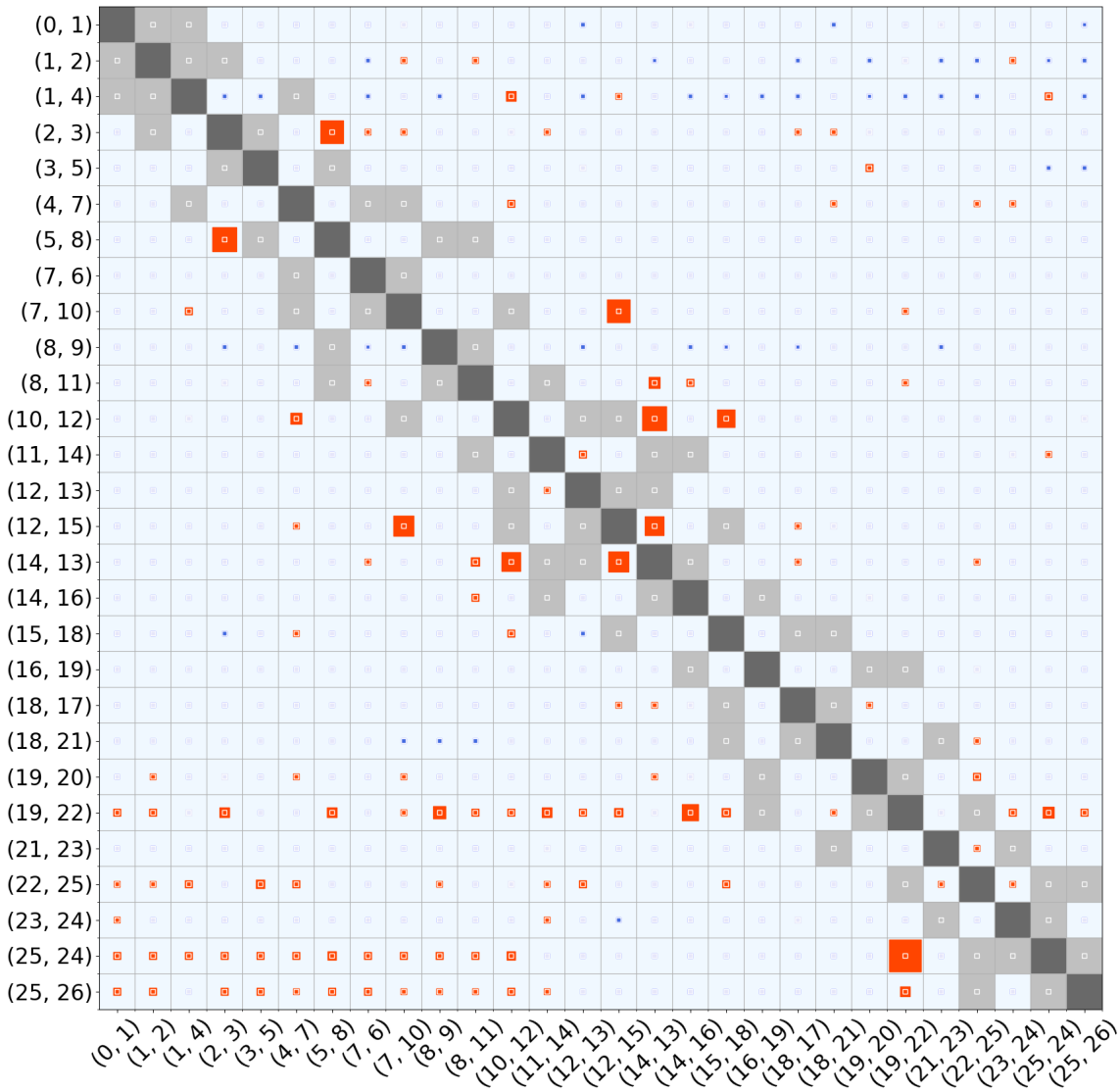


Figure 3.1: **Crosstalk distribution on the IBMQ Toronto system.** The independent cases are represented as small white boxes on the center of each cell whose size is fixed. When the conditional error rate is higher, the ratio is denoted as the size of orange boxes. On the contrary, it gets lower, represented as the size of blue boxes. The diagonal and cells next to diagonals filled with light gray indicate the combinations that cannot be executed in parallel.

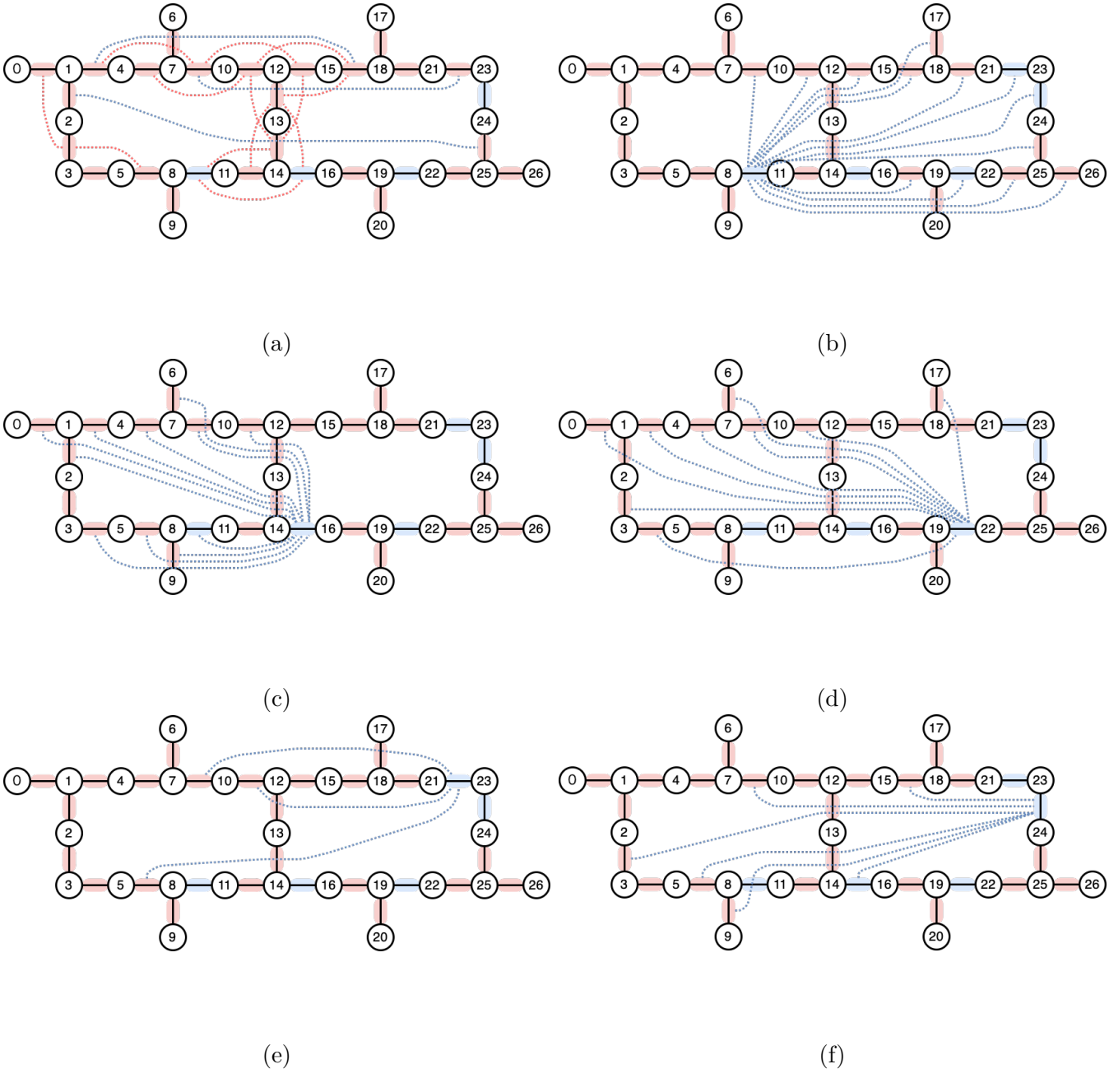


Figure 3.2: **Crosstalk distribution on** IBM Q System These figures show the crosstalk relationship on IBM Q System 27-qubit processor (IBM Q Toronto). The graph is a schematic representation of the processor, with each vertex representing a qubit and each edge representing a two-qubit coupling. Among the edges in the graph, the two-qubit coupling with crosstalk is indicated by the dashed connection. The red dashed lines represent crosstalk at a distance of one hop, and the blue dashed lines represent crosstalk more than two hops. The data show that most of the strong crosstalk occurs within one hop, and is particularly concentrated in the center of the processor 3.2a. On the other hand, certain two-qubit gates $((8, 11), (14, 16), (19, 22), (21, 23)$, and $(23, 24))$ share the crosstalk in long-range 3.2b–3.2f.

3.1.3 Reducing detection overhead strategy

In previous work on the noise-aware compilation, the software uses calibration data such as the gate error rates and coherence time provided on a daily basis to optimize the circuit. However, the execution time required to investigate crosstalk of all the gate combinations even for a 27-qubit system is not realistic [48]. Therefore, in this study, we limited the investigation items to the practical parts by following the procedure to reduce the computational cost of crosstalk detection.

Step 1: Randomly investigate the crosstalk relation. Since CNOT_j gates that are in a crosstalk noise relationship with a CNOT_i gate tend to be in a crosstalk noise relationship with other CNOT_k gates, for example $(19, 22)$ shown in Fig. 3.1 and Fig. 3.2d. First, we randomly select a few qubit pairs and investigate the existence of crosstalk. If there are any qubit pairs with crosstalk, we examine other combinations of them. By repeating this process to some extent, we can efficiently trace the error information of the entire processor.

Step 2: Investigate only strong crosstalk is observed. As shown in Fig. 3.3, crosstalk noises tend to appear in the same part of the processor. This is probably because crosstalk is caused by the design and packaging of the processor. Therefore we investigate only those parts that have particularly strong crosstalk errors obtained in steps 1 on daily usage.

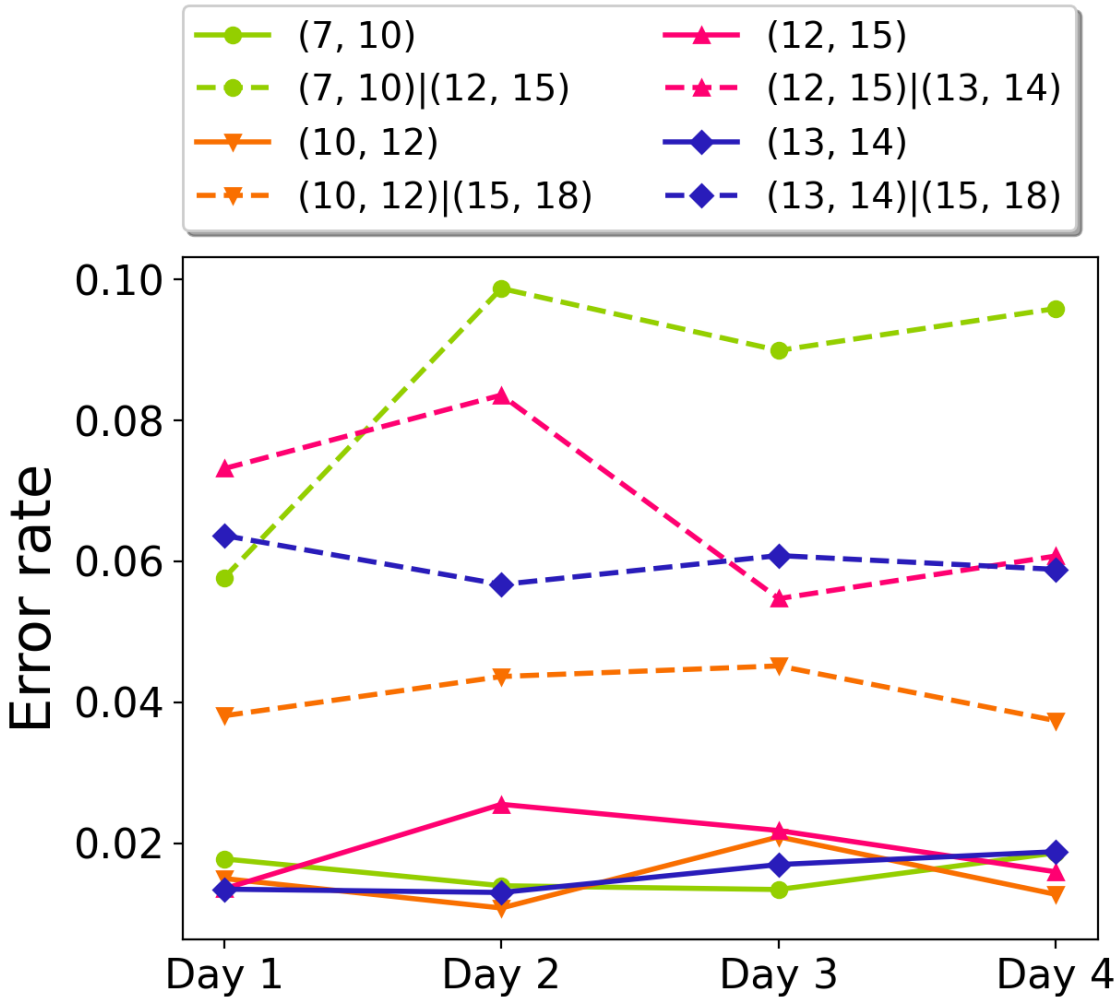


Figure 3.3: **Temporal variation in crosstalk noise.** The solid lines represent the independent error rate of two qubits gate, and the dashed lines denote the conditioned error rate when two $CNOT$ gates on different processor positions are executed concurrently. Conditioned error rates are 2x to 5x higher than the independent cases and tend to keep the ratio from its baseline for most periods.

3.2 Measurement Waiting Duration

3.2.1 Experimental setup

To evaluate the impact of measurement waiting duration, we conducted a preliminary experiment as shown in Fig. 3.4. We compare the output reliability between two circuits of several initial states, the one inserts waiting duration to the quantum circuit before gate operation and the other one after gate operation that may cause more decoherence effect. Here we use the Qiskit new command, Delay operation[53] to insert waiting duration to the quantum circuit.

(a) Delay duration before operation (Keep $|0\rangle$ state as late as possible.)

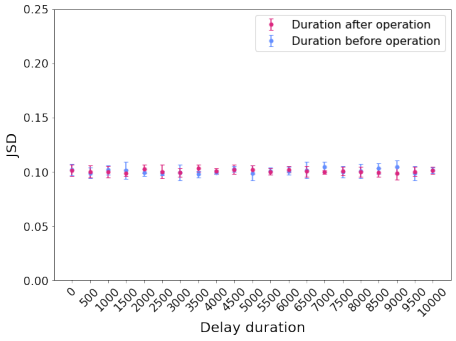
(b) Delay duration after operation U (Operate gate state as soon as possible).

Figure 3.4: **Circuits for delay insertion experiments.** To evaluate how impact the measurement waiting duration, the author compares two circuits. Upper one is inserted the delay duration before gate operation U. Another one inserts delay after operation that expected to cause higher decoherence error. In this experiments we use the Id gate, X gate and H gate in U part to initialize the qubit state.

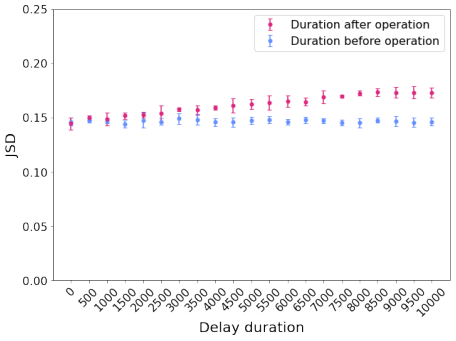
3.2.2 Impact of measurement waiting duration

As shown in Fig. 3.5, the output reliability vary The vertical axis denotes the Jensen-Shannon divergence between ideal probability distribution calculated by Qiskit Aer simulator and experimental result from IBM Q Toronto. That represents how close the experimental results to ideal. When JSD is 0, the two distribution are same. The horizontal axis represents delayed duration time inserted each experiments. dt is the units of duration time which is defined as $\frac{2}{9}$ns. The left column represents the result of $|0\rangle$ state case. The middle column are initialized as $|1\rangle$ state by X gate and the right ones are prepared as $|+\rangle$ state by H gate. To make the execution time the same, we use Id gate in $|0\rangle$ state case. The maximum delay duration here 10×10^6 dt is approximately double the time of T_2 qubit life time of IBM Q Toronto. We execute 5 times of 8192 trial and all error bar represents 3σ bounds. The $|0\rangle$ case shows that

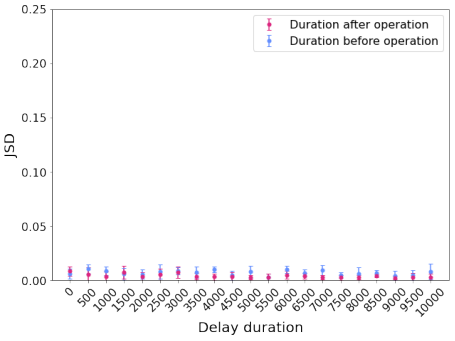
keeping ground state as late as possible (ALAP) may mitigate the decoherence effect of the qubit state.



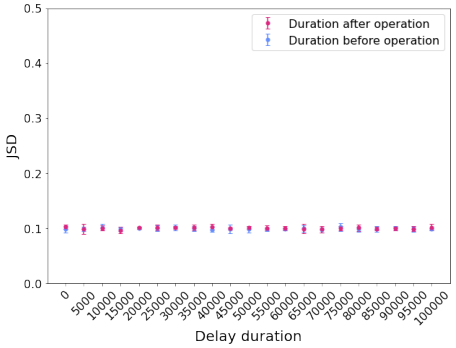
(a) $|0\rangle$ state, from 0 to 10×10^4 dt



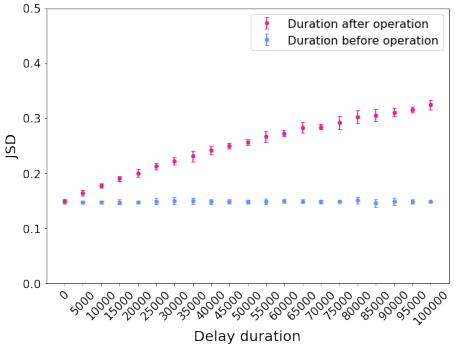
(b) $|1\rangle$ state from 0 to 10×10^4 dt



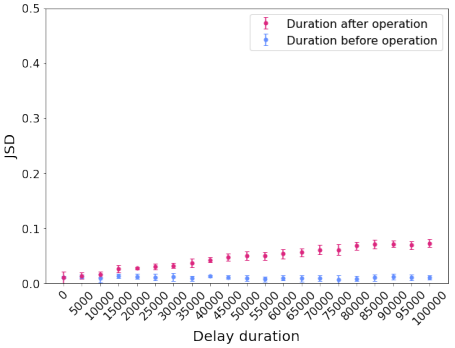
(c) $|+\rangle$ state from 0 to 10×10^4 dt



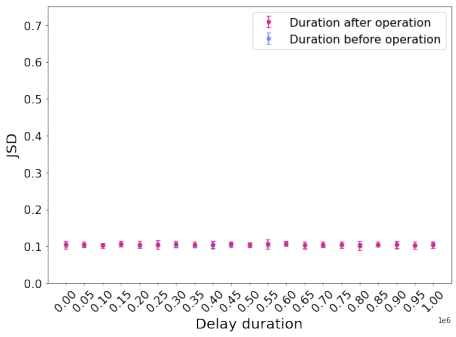
(d) $|0\rangle$ state from 0 to 10×10^5 dt



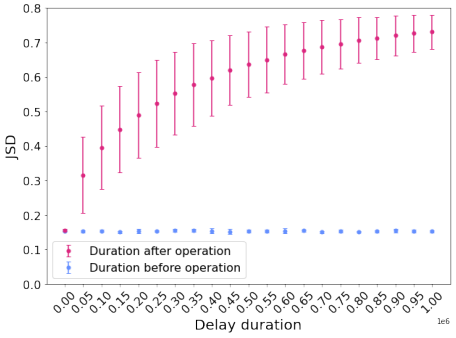
(e) $|1\rangle$ state from 0 to 10×10^5 dt



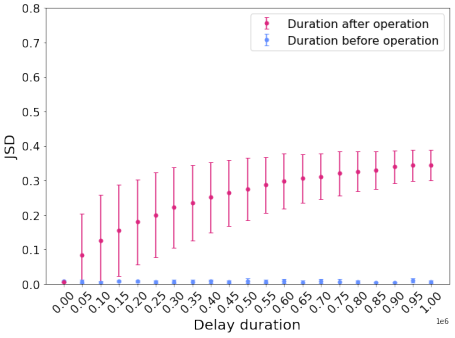
(f) $|+\rangle$ state from 0 to 10×10^5 dt



(g) $|0\rangle$ state from 0 to 10×10^6 dt



(h) $|1\rangle$ state from 0 to 10×10^6 dt



(i) $|+\rangle$ state from 0 to 10×10^6 dt

Figure 3.5: **The impact of measurement waiting duration on three initial state.**

Chapter 4

Proposed Algorithms

4.1 Qubit allocation policy

The software developed in this study takes multiple programmed quantum circuits as input and combines them into a single compiled quantum circuit as output. In the compilation phase, the initialization of physical qubits is an important issue of NISQ computing, like a reduction of $SWAP$ gates. In particular, when multiple independently programmed quantum circuits are executed on a processor simultaneously, qubit assignment that takes crosstalk noise into account can minimize interference with independent circuits and improve the reliability of the entire quantum computation.

4.1.1 High cost circuit first allocation

Our multi-programming layout method is implemented as an extension of Noise-adaptive layout [31], [54] used in the current Qiskit transpiler.

At first, the muli-compiler analyzes the error information of quantum chip based on $CNOT$ and measurement error rates as a weighted graph $G(V, E)$, here V: vertex is the number of qubits, E: edge is the number of qubits connection and the weight is the reliability of qubits or edges, same as Noise-adaptive layout method [54]. Then in the mapping phase, it compares the cost of the individual quantum circuit in a simultaneous execution set, that represents reliability calculated as a number of qubits times $CNOT$ depth. And then search for better allocation like the most costly program to the most reliable physical qubits based on greedy heuristics.

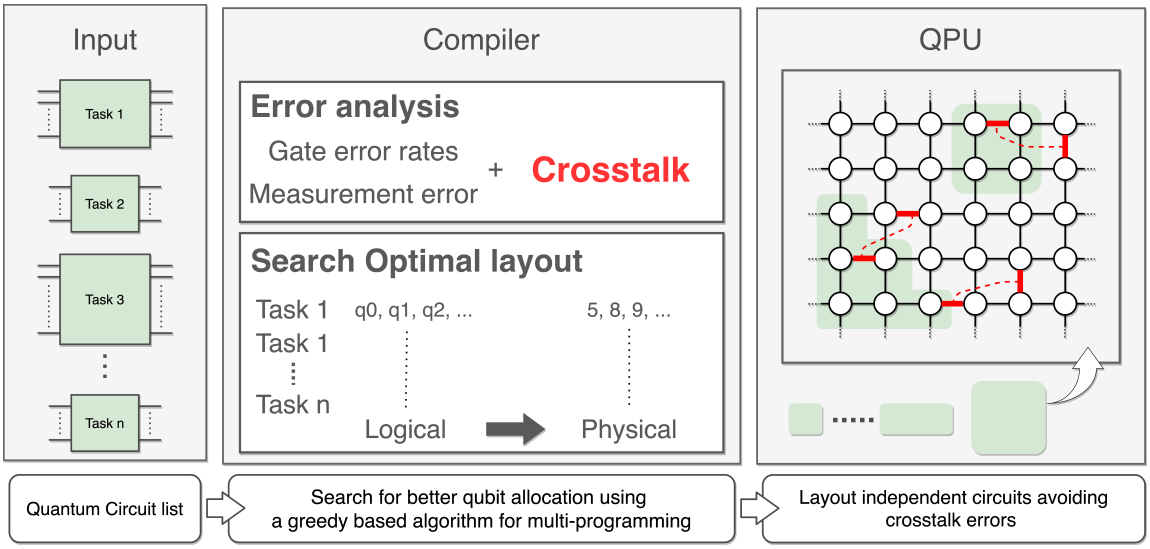


Figure 4.1: **Compilation procedure of multi-programming circuit** Here we show our proposal compilation procedure for parallel execution of multiple quantum circuits. As the input, the compiler takes a list of quantum circuits to be executed on same round. To prioritize physical qubits allocation of independent programs in parallel execution, we defined the cost $\mathcal{C}$ the product of the number of qubits and the number of $CNOT$ gates. The circuits are allocated to physical qubits from the higher cost first. In the allocation phase, the most reliable qubits are searched and allocated based on error information (gate error, measurement error, crosstalk) of the processor using the greedy method.

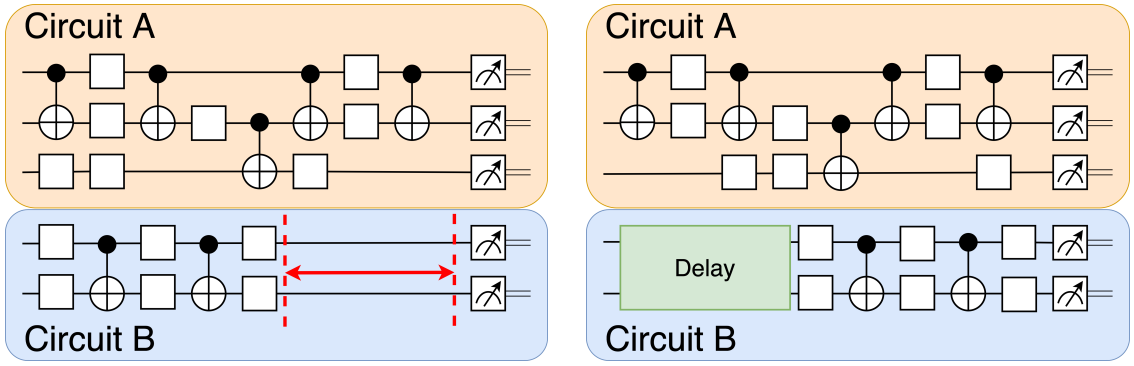
4.1.2 Crosstalk aware qubit allocation

For the crosstalk mitigation, we extend our multi-compiler by using crosstalk noise data taken from the processor, in addition to gate and measurement error rates that are originally used. Every time program instruction allocated to physical qubits, the compiler re-analyze the noise graph of processor based on the crosstalk error on that physical qubits that is introduced in Sec. 3.1.

4.2 Scheduling multi-programming instruction policy

As shown in Sec. 3.2, keeping qubit in the ground state can mitigate decoherence error. Current IBM Q System except for Hummingbird r2, measurement operation is performed on all hardware qubits at once as a trigger. In the multi-programming case, during waiting for longer circuit's gate operations and measurement, the shorter program may decohere unnecessarily Fig. 4.2a. To mitigate this error, we propose a scheduling method, *multi-ALAP scheduling* method as an extension of Qiskit scheduler, ALAP pass [55] similar to the delayed instruction method proposed in [4]. However, the current processor's qubit lifetime is much better than when previous work conducted. We verify how this type of scheduling method works on the current IBM Q System in Sec 5.3.

Multi-ALAP method schedules each gate in the circuit as late as possible to keep each qubit in the ground state Fig. 4.2b.



(a) As soon as possible schedule.

(b) As late as possible schedule

Figure 4.2: **Scheduling for decoherence mitigation in the multi-programming case.**

Chapter 5

Experiments and Evaluation

We conducted two series of evaluations for our proposals (Crosstalk adaptive layout, Multi-ALAP scheduling). In the entire experiment, we care about the output reliability as JSD of output distributions. As a baseline, we compare those to the Qiskit transpile method.

5.1 Setup

We use small size benchmark circuits from previous work [56] as shown in Table 5.1.

We used IBM Quantum Experience API [2], IBM Q Toronto and Qiskit version 0.23.1, Qiskit ignis version 0.5.1, Qiskit terra version0.16.1 Intel Core i5 processor (1.6GHz, 4GB RAM), Python version 3.8.1 to evaluate proposed method and compared to the qiskit transpiler with Dense layout, Noise-Adaptive layout [31], SABRE layout [57].

To quantify the performance of proposed method, we used Jsensen-Shannon Divergence (JSD). JSD is the metric for comparing the distance of two probability distributions. When distributions are same, JSD is 0. The definition of JSD contains Kullback-Liebler Divergence. KLD is defined as following:

$$D(p||q) = \sum_i p(i) \ln \frac{p(i)}{q(i)} \tag{5.1}$$

and JSD is defined as following:

$$m(i) = \frac{1}{2}(p(i) + q(i)) \tag{5.2}$$

Benchmark	Description	Qubits	Gates	CX
deutsch	Deutsch algorithm with 2 qubits for $f(x) = x$	2	5	1
grover	Grover's algorithm	2	16	2
linearsolver	Solver for a linear equation of one qubit	3	19	4
toffoli	Toffoli gate	3	18	6
fredkin	Fredkin gate	3	19	8
adder	Quantum ripple-carry adder	4	23	10
error_correctiond3	Error correction with distance 3 and 5 qubits	5	114	49

Table 5.1: **Small Benchmark Circuits.** We picked several small size quantum circuit to benchmark our proposal from benchmark circuits set called QASMBench [56].

$$\mathrm{JSD}(p, q) = [\frac{1}{2}D(p||m) + \frac{1}{2}D(q||m)]^{\frac{1}{2}} \tag{5.3}$$

where p and q represent each probability distributions and i corresponds to a possible item of them.

5.2 Crosstalk adaptive layout

In the first series, we evaluate the performance of our new qubit allocation policy that is called Crosstalk adaptive layout, the enhanced version of the Noise adaptive layout method in Qiskit [31]. We compared our proposed layout method to the Qiskit transpiler [58]. We used several combinations of quantum circuits from QASMBench [56] shown in Table 5.2 and Table 5.3. At first, the compiler converts a set of circuits to one composed circuit and allocates it to the hardware qubits by applying each layout method. And then execute it as a multi-programming on IBM Q System. To quantify the noise effect to the reliability of parallel execution, we also run circuits set independently that allocated the same hardware qubits as the multi-programming case and compare those output quality by using Jensen-Shannon Divergence (JSD) introduced in Sec. 5.1.

5.2.1 Results

To evaluate the output reliability of the layout method, we use Jensen-Shannon divergence (JSD) between output distributions. From the definition, the lower JSD value represents better results in this case. Fig. 5.1 shows the results of the parallel

execution of 3 circuits case, and Fig. 5.2 4 circuits case. Here, we compare our crosstalk adaptive approach to Qiskit layout passes, dense layout, noise-adaptive layout, and SABRE layout.

Our proposal outperforms the baselines for some of the benchmark circuit combinations 3, 4 and 5 cases shown in Table 5.2, Table 5.3 and Table 5.4. Particularly our method works well in the case of the quantum circuits with 4 to 10 $CNOT$ gates, allocated later which means the program assigned less reliable qubits in the processor. On the contrary, in the shorter operation circuits case with few $CNOT$ gates that includes deutsch, grover, linear solver , IBM Q Toronto can treat each task with high reliability regardless of the choice of the layout method. And also, in the case of the longer operation circuit case with tens of $CNOT$ gates, error_correctiond3, it seems to be difficult to process high fidelity in current processors. This shows our techniques avoid the unwanted decaying caused by parallel processing and improve the utilization of the processor efficiently.

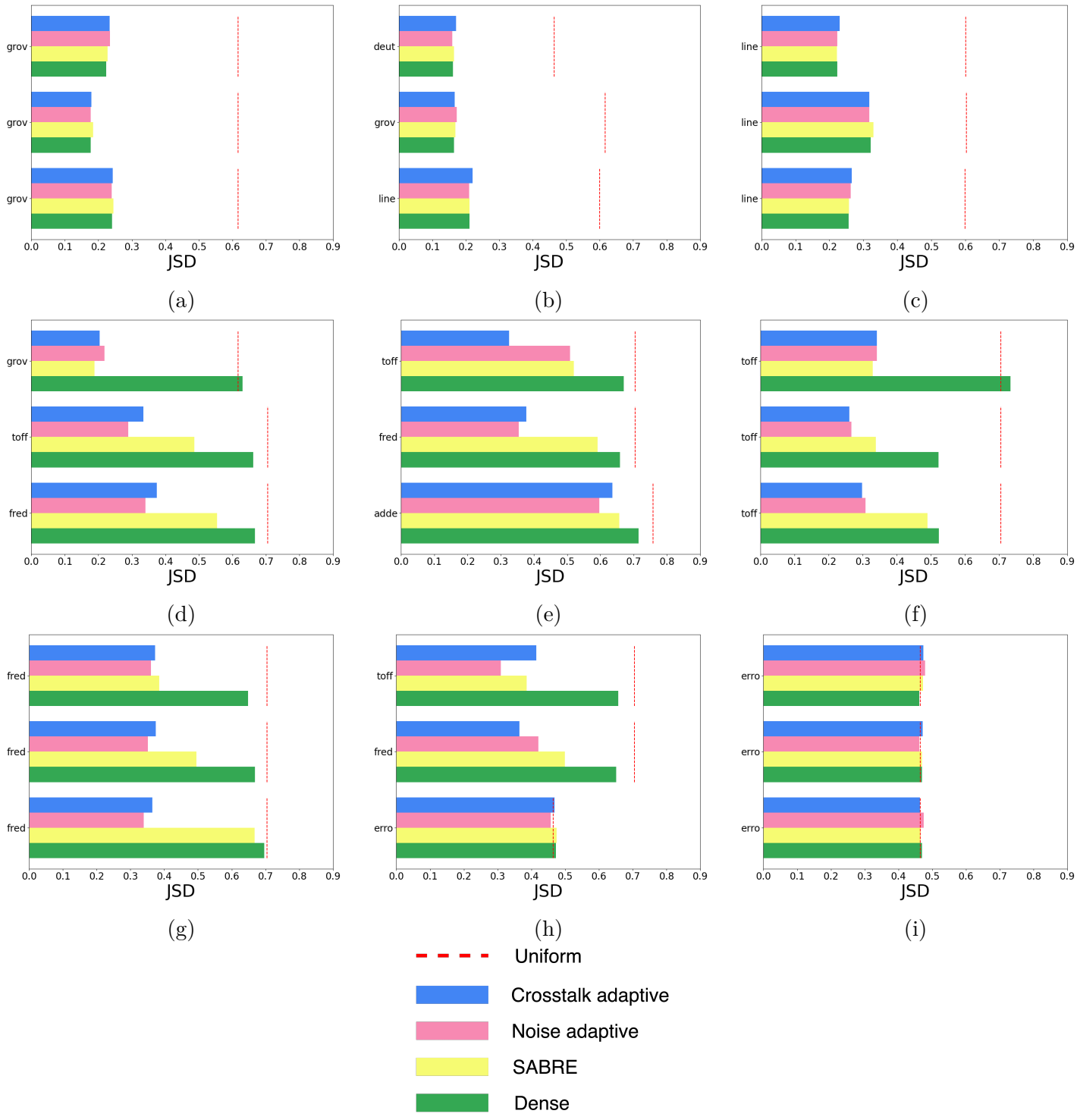


Figure 5.1: **Comparison of JSD of the parallel execution consisting 3 circuits case.** We compared our proposal layout method Crosstalk-adaptive (Blue) to Qiskit compilation methods include Dense layout(Green), Noise-adaptive layout(Red), SABRE layout(Yellow). Each sub-figures are the benchmark case that corresponds to Table 5.2. For example, 5.1a is one multi-programming circuit consisting **fredkin**, **toffoli** and **grover** circuit shown in the first row in Table 5.2. JSD shows how close the result from IBM Q Toronto to the ideal probability distribution calculated by Qiskit Aer simulator. From the definition, when JSD is 0, two distributions are the same.

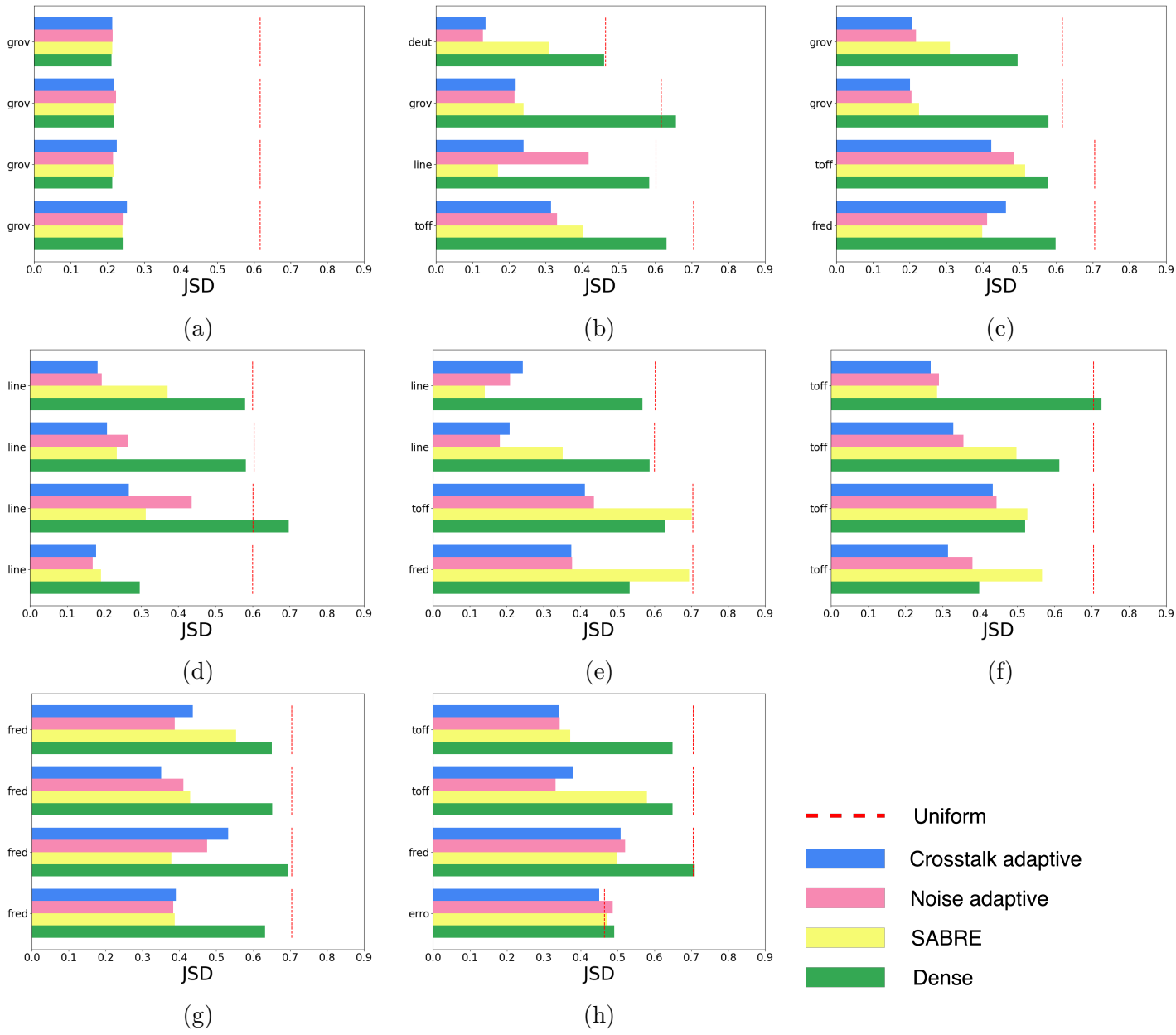


Figure 5.2: **Comparison of JSD of the parallel execution consisting 4 circuits case.** We compared our proposal layout method Crosstalk-adaptive (Blue) to Qiskit compilation methods include Dense layout(Green), Noise-adaptive layout(Red), SABRE layout(Yellow). Each sub-figures are the benchmark case that corresponds to Table 5.3. For example, 5.2d is one multi-programming circuit consisting two **adder**, **fredin** and **toffoli** circuits shown in the first row in Table 5.3. JSD shows how close the result from IBM Q Toronto to the ideal probability distribution calculated by Qiskit Aer simulator. From the definition, when JSD is 0, two distributions are the same.

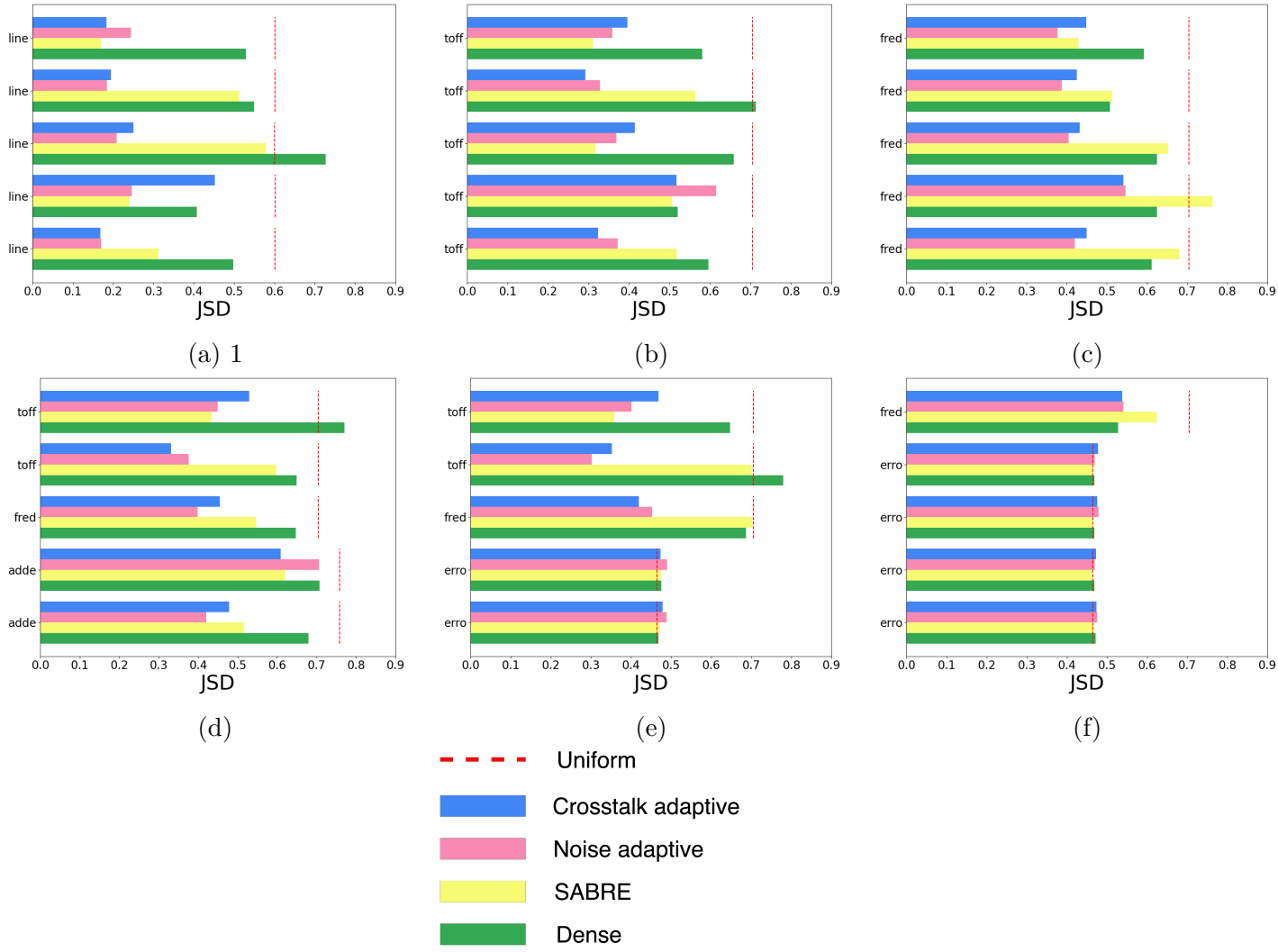


Figure 5.3: **Comparison of JSD of the parallel execution consisting 5 circuits case.** We compared our proposal layout method Crosstalk-adaptive (Blue) to Qiskit compilation methods include Dense layout(Green), Noise-adaptive layout(Red), SABRE layout(Yellow). Each sub-figures are the benchmark case that corresponds to Table 5.4. For example, 5.3d is one multi-programming circuit consisting two **adder**, **fredkin** and two **toffoli** circuits shown in the fourth row in Table 5.4. JSD shows how close the result from IBM Q Toronto to the ideal probability distribution calculated by Qiskit Aer simulator. From the definition, when JSD is 0, two distributions are the same.

5.3 Modifying Measurement Waiting Duration

In the second series, we evaluate the performance of Multi-ALAP scheduling method introduced in Sec. 4.2. To evaluate the proposed method, we compare it to the output of non-optimized scheduling cases. Fig. 5.4, Fig. 5.5, and Fig. 5.6 show the impact of optimized scheduling of each number of circuit set of multi-programming cases correspond to Table 5.2, Table 5.3, and Table 5.4.

5.3.1 Evaluation with benchmark circuits

In the horizontal axis, each bar represents JSD that shows how close the result from IBM Q System to the ideal probability distribution calculated by a noise-free simulation with Qiskit Aer simulator. From the definition of JSD, 0 is the ideal case. The red dashed line denotes the JSD between ideal results and uniform distribution that corresponds to a noisy random output case. Similar to the previous study [4] used IBM Q 16 Melbourne, we found our proposed method outputs almost the same results as the non-optimized case on the current leading processor IBM Q Sydney 27-qubit system. We assume the qubit coherent time is much longer than the tolerable circuit depth on the current IBM Q System.

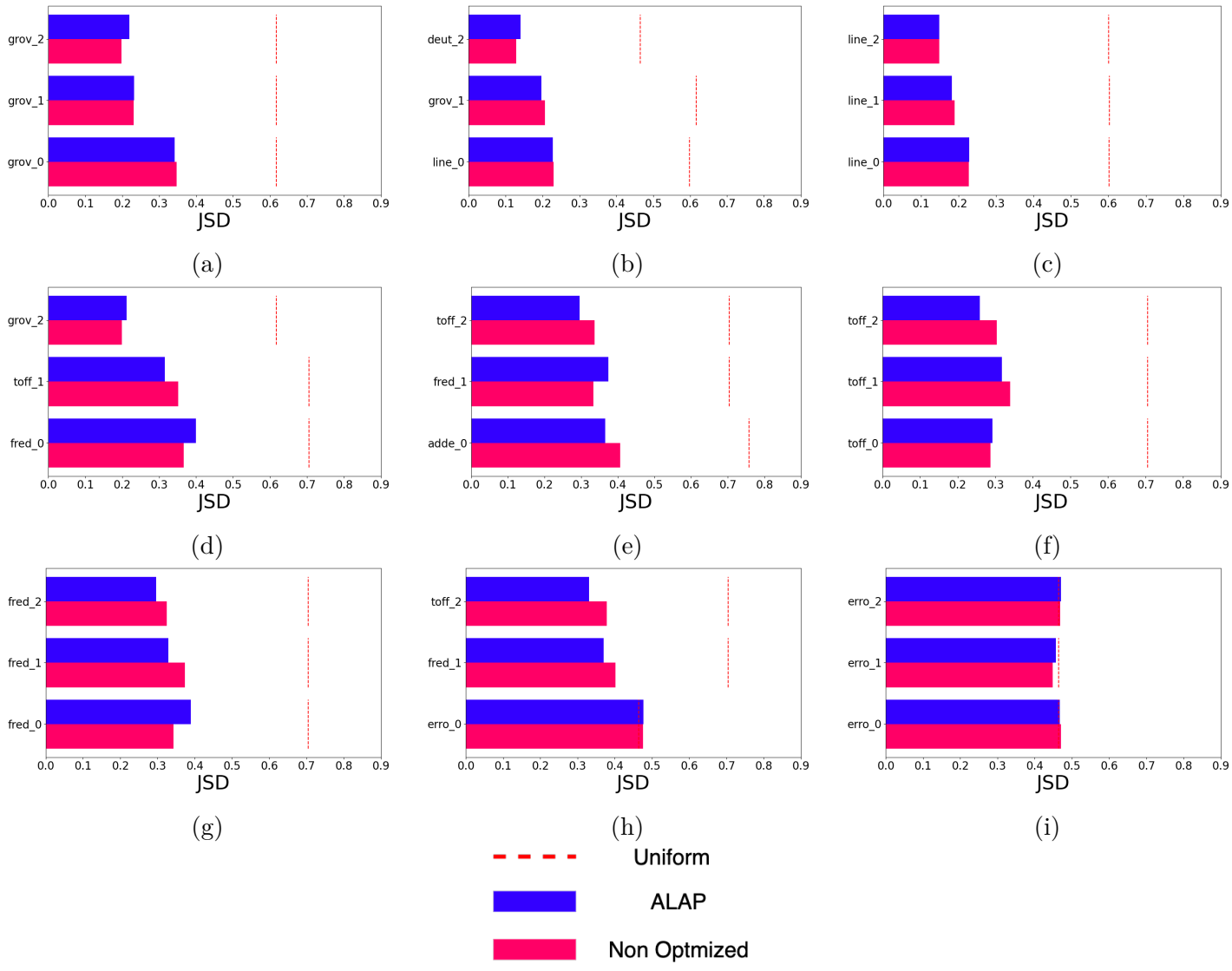


Figure 5.4: **JSD of Multi-ALAP scheduling: 3 circuits multi-programming case.** We compared our proposal scheduling Multi ALAP scheduling (Blue) to non-optimized (Red). Each sub-figures are the benchmark case that corresponds to Table 5.2. For example, 5.4b is one multi-programming circuit consisting **linearsolver**, **grover** and **deutsch** circuit shown in the second row in Table 5.2. JSD shows how close the result from IBM Q Toronto to the ideal probability distribution calculated by Qiskit Aer simulator. From the definition, when JSD is 0, two distributions are the same.

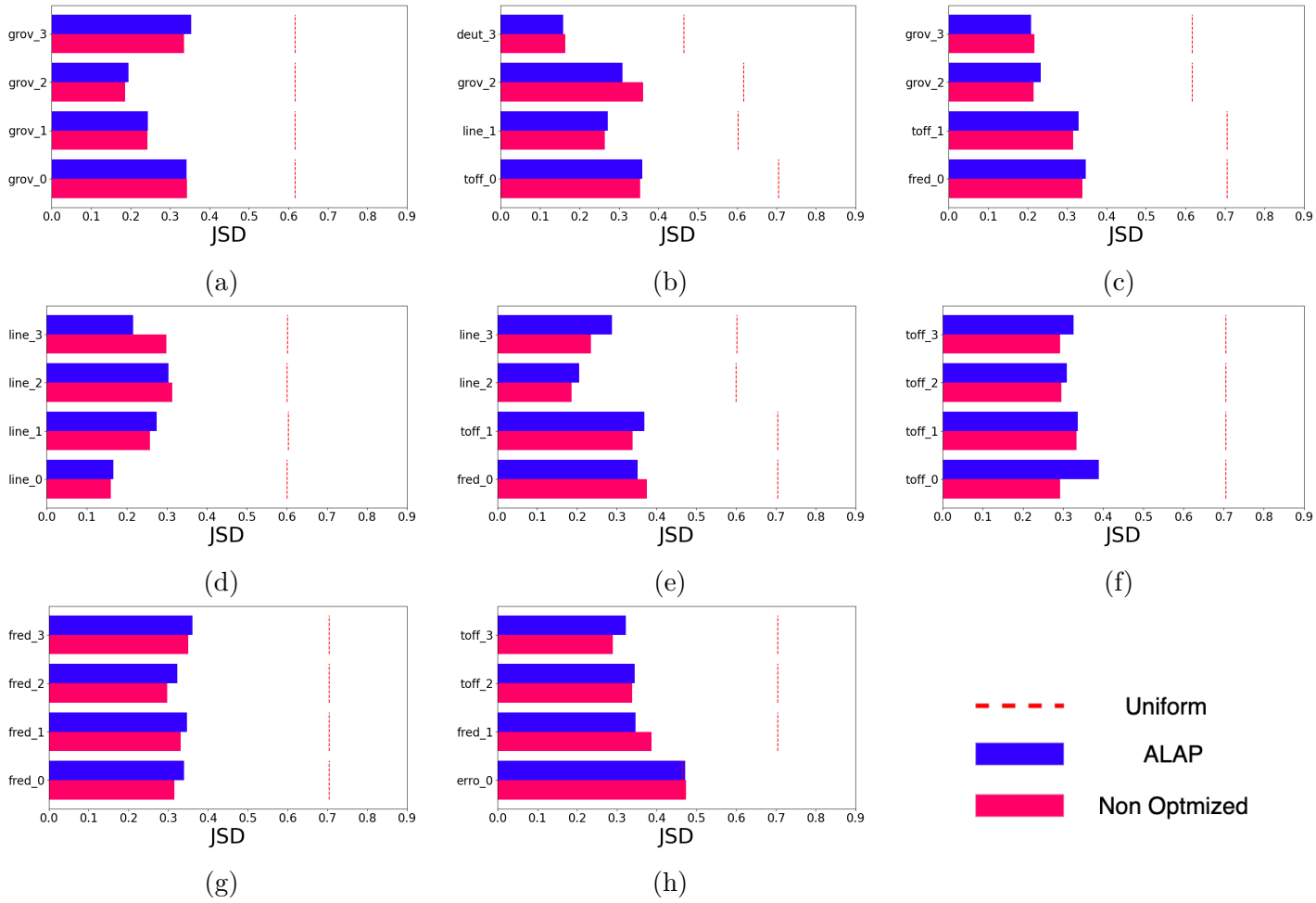


Figure 5.5: **JSD of Multi-ALAP scheduling: 4 circuits multi-programming case.** We compared our proposal scheduling Multi ALAP scheduling (Blue) to non-optimized (Red). Each sub-figures are the benchmark case that corresponds to Table 5.3. For example, 5.5b is one multi-programming circuit consisting **toffoli**, **linearsolver**, **grover** and **deutsch** circuits shown in the second row in Table 5.3. JSD shows how close the result from IBM Q Toronto to the ideal probability distribution calculated by Qiskit Aer simulator. From the definition, when JSD is 0, two distributions are the same.

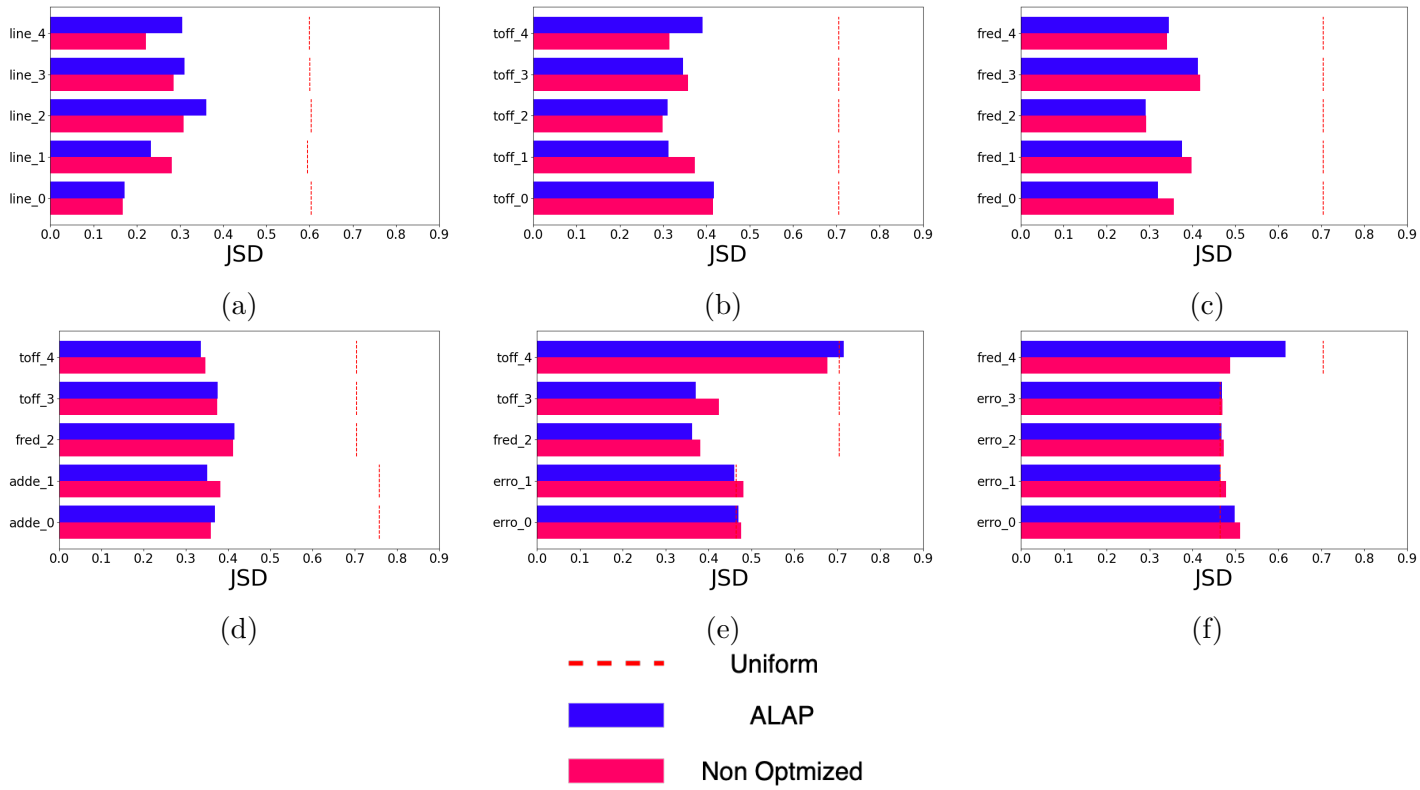


Figure 5.6: **JSD of Multi-ALAP scheduling: 5 circuits multi-programming case.** We compared our proposal scheduling Multi ALAP scheduling (Blue) to non-optimized (Red). Each sub-figures are the benchmark case that corresponds to Table 5.3. For example, 5.6d is one multi-programming circuit consisting two **adder**, **fredkin**, and two **toffoli** circuits shown in the fourth row in Table 5.3. JSD shows how close the result from IBM Q Toronto to the ideal probability distribution calculated by Qiskit Aer simulator. From the definition, when JSD is 0, two distributions are the same.

Label	Name			Qubits			Gates			CX		
(a)	grov	grov	grov	2	2	2	16	16	16	2	2	2
(b)	line	grov	deut	3	2	2	19	16	5	4	2	1
(c)	line	line	line	3	3	3	19	19	19	4	4	4
(d)	fred	toff	grov	3	3	2	19	18	16	8	6	2
(e)	adde	fred	toff	4	3	3	23	19	18	10	8	6
(f)	toff	toff	toff	3	3	3	18	18	18	6	6	6
(g)	fred	fred	fred	3	3	3	19	19	19	8	8	8
(h)	erro	fred	toff	5	3	3	114	19	18	49	8	6
(i)	erro	erro	erro	5	5	5	114	114	114	49	49	49

Table 5.2: 3 benchmark circuits multi-programming case.

Label	Name				Qubits				Gates				CX			
(a)	grov	grov	grov	grov	2	2	2	2	16	16	16	16	2	2	2	2
(b)	toff	line	grov	deut	3	3	2	2	18	19	16	5	6	4	2	1
(c)	fred	toff	grov	grov	3	3	2	2	19	18	16	16	8	6	2	2
(d)	line	line	line	line	3	3	3	3	19	19	19	19	4	4	4	4
(e)	fred	toff	line	line	3	3	3	3	19	18	19	19	8	6	4	4
(f)	toff	toff	toff	toff	3	3	3	3	18	18	18	18	6	6	6	6
(g)	fred	fred	fred	fred	3	3	3	3	19	19	19	19	8	8	8	8
(h)	erro	fred	toff	toff	5	3	3	3	114	19	18	18	49	8	6	6

Table 5.3: 4 benchmark circuits multi-programming case.

Label	Name					Qubits					Gates					CX				
(a)	line	line	line	line	line	3	3	3	3	3	19	19	19	19	19	4	4	4	4	4
(b)	toff	toff	toff	toff	toff	3	3	3	3	3	18	18	18	18	18	6	6	6	6	6
(c)	fred	fred	fred	fred	fred	3	3	3	3	3	19	19	19	19	19	8	8	8	8	8
(d)	adde	adde	fred	toff	toff	4	4	3	3	3	23	23	19	18	18	10	10	8	6	6
(e)	erro	erro	fred	toff	toff	5	5	3	3	3	114	144	19	18	18	49	49	8	6	6
(f)	erro	erro	erro	erro	fred	5	5	5	5	3	114	114	114	114	114	49	49	49	49	49

Table 5.4: 5 benchmark circuits multi-programming case.

Chapter 6

Conclusion and Remarks

Crosstalk-adaptive layout method cares about crosstalk characterization of the processor in the qubit allocation phase of the compilation process. To mitigate unnecessary decoherence, we proposed Multi-ALAP scheduling method that moves shorter circuits in multi-programming to a later schedule. Our evaluations show that two noise factors, crosstalk in current processor and difference of operational duration between independent QC in multi-programming, do not have a serious impact on the output reliability of parallel execution. In the case of the layout method, our proposal outputs a qubit allocation pattern that is similar to the Noise-adaptive layout method. We consider since the crosstalk errors in the current processor are sufficiently suppressed, our proposal could not make a different result. Future work is required in order to verify these results comes from the improvement of the processor itself, which conduct further experiments on old model processors to compare the results. For the scheduling method, we consider the qubit coherent time of the current processor is enough for the executable quantum circuits that only contain around 10 $CNOT$ gates due to the gate error rates. Except for **error_correctiond3**, the operational duration of our benchmarks finished about 40 % of average T_1 time coherent time of IBM Q Toronto. We expect our scheduling method performs well on the processor that can accept the longer depth of quantum circuit with smaller gate error rates or when the difference of operational durations of independent circuits in multi-programming is bigger.

Acknowledgement

I would like to thank my supervisor, Professor Rodney Van Meter. I have been in his lab for three years since I was a sophomore. He gave me not only the method of research but also the attitude of research and various lessons. I thank him for allowing me to present at conferences and write dissertations. Thanks to his financial support for the participation in wonderful hackathon events, including, QiskitCamp2019, QiksitCampAsia2019. I would like to thank my supervisor, Project Assistant Professor Takahiko Satoh. He worked with me on almost every project I worked on. He carefully discussed each of my childish proposals for research, provided theoretical support and new perspectives. Thanks to Jun Murai. He taught me a serious attitude towards the Internet and technology.

Thanks to my senior Takaaki Matsuo. He showed me how to work in university life and lab activities, and consulted me many times, and provided appropriate advice. Thanks to Ryosuke Satoh. I grew up working with him for Mitou Target and NQC project for almost 3 years in total. Ryosuke also strongly supported my activities at the laboratory and taught me too many things. Thanks to Ryusei Siiba, Takanori Hirano, and Shigetora Miyashita. Talking with them in the lab always gave me witty stories, and I was able to spend a fun time. Thanks to Takuma Nagaoka. It was great for me that Takuma gave me many questions and advice from a keen point of view. Thanks to Shigetora Miyashita, Makoto Nakai, Sitong Liu, Yinjie Zhou, Nozomi Tanetani, Kosuke Onishi, Takuma Nagaoka, Hikaru Yokomori, Ryota Kikuchi, and Taiga Yanagida for their active contribution to AQUA.

Thanks to the members of the IBMQ team, Qiskit Community. Thanks to them, I was able to conduct various experiments and learn.

Finally, I thank the Faculty and all members of the Tokuda, Murai, Kusumoto, Nakamura, Takashio, Van Meter, Uehara, Mitsugi, Nakazawa, Takeda Joint Research Group (RG) and my family for all the supports.

Bibliography

[1] John Preskill. *Quantum Computing in the NISQ era and beyond*. 2018. DOI: 10.22331/q-2018-08-06-79. arXiv: 1801.00862.

[2] IBM. *IBM Quantum Experience*. https://quantum-computing.ibm.com/.

[3] Rigetti. *Rigetti Quantum Cloud Services*. https://qcs.rigetti.com/.

[4] Poulami Das et al. "A case for multi-programming Quantum computers". In: *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*. IEEE Computer Society, Oct. 2019, pp. 291–303. ISBN: 9781450369381. DOI: 10.1145/3352460.3358287.

[5] Lei Liu and Xinglei Dou. *A New Qubits Mapping Mechanism for Multi-programming Quantum Computing*. 2020. arXiv: 2004.12854 [cs.DC].

[6] E. Knill et al. "Randomized benchmarking of quantum gates". In: *Physical Review A - Atomic, Molecular, and Optical Physics* 77.1 (2008), pp. 1–7. ISSN: 10502947. DOI: 10.1103/PhysRevA.77.012307. arXiv: 0707.0963.

[7] J. M. Chow et al. "Randomized benchmarking and process tomography for gate errors in a solid-state qubit". In: *Physical Review Letters* 102.9 (2009), pp. 1–4. ISSN: 00319007. DOI: 10.1103/PhysRevLett.102.090502. arXiv: 0811.4387.

[8] Kenneth Rudinger et al. "Probing Context-Dependent Errors in Quantum Processors". In: 021045 (2019), pp. 1–12. DOI: 10.1103/PhysRevX.9.021045.

[9] Pranav Mundada et al. "Suppression of Qubit Crosstalk in a Tunable Coupling Superconducting Circuit". In: *Phys. Rev. Applied* 12 (5 Nov. 2019), p. 054023. DOI: 10.1103/PhysRevApplied.12.054023. URL: https://link.aps.org/doi/10.1103/PhysRevApplied.12.054023.

[10] David C. McKay et al. "Three-Qubit Randomized Benchmarking". In: *Physical Review Letters* 122.20 (2019), pp. 1–6. ISSN: 10797114. DOI: 10.1103/PhysRevLett.122.200502. arXiv: 1712.06550.

[11] P. Krantz et al. “A quantum engineer’s guide to superconducting qubits”. In: *Applied Physics Reviews* 6.2 (2019). ISSN: 19319401. DOI: 10.1063/1.5089550. arXiv: 1904.06560.

[12] IBM Research. *IBM’s Roadmap For Scaling Quantum Technology*. https://www.ibm.com/blogs/research/2020/09/ibm-quantum-roadmap/.

[13] Google AI. *A Preview of Bristlecone, Google’s New Quantum Processor*. https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html.

[14] Peter W Shor. “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer”. In: *SIAM review* 41.2 (1999), pp. 303–332.

[15] Lov K Grover. “A fast quantum mechanical algorithm for database search”. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, pp. 212–219.

[16] Daniel Gottesman. “Stabilizer codes and quantum error correction”. In: *arXiv preprint quant-ph/9705052* (1997).

[17] Austin G Fowler et al. “Surface codes: Towards practical large-scale quantum computation”. In: *Physical Review A* 86.3 (2012), p. 032324.

[18] Kishor Bharti et al. *Noisy intermediate-scale quantum (NISQ) algorithms*. 2021. arXiv: 2101.08448 [quant-ph].

[19] Yudong Cao et al. “Quantum Chemistry in the Age of Quantum Computing”. In: *Chemical Reviews* 119.19 (Aug. 2019), pp. 10856–10915. ISSN: 1520-6890. DOI: 10.1021/acs.chemrev.8b00803. URL: http://dx.doi.org/10.1021/acs.chemrev.8b00803.

[20] Suguru Endo et al. *Hybrid quantum-classical algorithms and quantum error mitigation*. 2020. arXiv: 2011.01382 [quant-ph].

[21] Suguru Endo, Iori Kurata, and Yuya O. Nakagawa. “Calculation of the Green’s function on near-term quantum computers”. In: *Phys. Rev. Research* 2 (3 Aug. 2020), p. 033281. DOI: 10.1103/PhysRevResearch.2.033281. URL: https://link.aps.org/doi/10.1103/PhysRevResearch.2.033281.

[22] Alberto Peruzzo et al. “A variational eigenvalue solver on a photonic quantum processor”. In: *Nature Communications* 5.1 (July 2014). ISSN: 2041-1723. DOI: 10.1038/ncomms5213. URL: http://dx.doi.org/10.1038/ncomms5213.

[23] Jarrod R McClean et al. “The theory of variational hybrid quantum-classical algorithms”. In: *New Journal of Physics* 18.2 (Feb. 2016), p. 023023. ISSN: 1367-2630. DOI: 10.1088/1367-2630/18/2/023023. URL: http://dx.doi.org/10.1088/1367-2630/18/2/023023.

[24] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. *A Quantum Approximate Optimization Algorithm*. 2014. arXiv: 1411.4028 [quant-ph].

[25] Takahiko Satoh, Yasuhiro Ohkura, and Rodney Van Meter. “Subdivided Phase Oracle for NISQ Search Algorithms”. In: *IEEE Transactions on Quantum Engineering* 1 (2020), pp. 1–15. ISSN: 2689-1808. DOI: 10.1109/tqe.2020.3012068. URL: http://dx.doi.org/10.1109/TQE.2020.3012068.

[26] Robert Sutor. *Dancing with Qubits*. Nov. 2019. ISBN: 978-1-83882-736-6.

[27] IBM. *Defining Quantum Circuits*. https://qiskit.org/textbook/ch-algorithms/defining-quantum-circuits.html.

[28] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. 10th. USA: Cambridge University Press, 2011. ISBN: 1107002176.

[29] 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.

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

[31] Prakash Murali et al. “Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers”. In: *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS* (2019), pp. 1015–1029. DOI: 10.1145/3297858.3304075. arXiv: 1901.11054.

[32] Shin Nishio et al. “Extracting Success from IBM ’ s 20-Qubit Machines Using Error-Aware Compilation”. In: 16.3 (2020).

[33] Matteo G. Pozzi et al. *Using Reinforcement Learning to Perform Qubit Routing in Quantum Compilers*. 2020. arXiv: 2007.15957 [quant-ph].

[34] Amazon. *Amazon Braket*. https://aws.amazon.com/jp/braket/.

[35] D-wave. *D-wave Leap*. https://cloud.dwavesys.com/leap/.

[36] Google. *Quantum Computing Playground*. https://experiments.withgoogle.com/quantum-computing-playground.

[37] Microsoft. *Microsoft Azure Quantum*. https://azure.microsoft.com/en-us/services/quantum/.

[38] Rigetti. *Rigetti Forest*. https://www.rigetti.com/quantum-computing/.

[39] Xanadu. *Xanadu Qauntum Cloud*. https://www.xanadu.ai/.

[40] Frank Leymann et al. *Quantum in the Cloud: Application Potentials and Research Opportunities*. 2020. arXiv: 2003.06256 [quant-ph].

[41] Jay M. Gambetta et al. "Characterization of addressability by simultaneous randomized benchmarking". In: *Physical Review Letters* 109.24 (Dec. 2012). ISSN: 00319007. DOI: 10.1103/PhysRevLett.109.240504. arXiv: 1204.6308.

[42] Sarah Sheldon et al. "Procedure for systematically tuning up cross-talk in the cross-resonance gate". In: *Physical Review A* 93.6 (2016), pp. 1–5. ISSN: 24699934. DOI: 10.1103/PhysRevA.93.060302. arXiv: 1603.04821.

[43] P. Krantz et al. "A quantum engineer's guide to superconducting qubits". In: *Applied Physics Reviews* 6.2 (June 2019), p. 021318. ISSN: 1931-9401. DOI: 10.1063/1.5089550. URL: http://dx.doi.org/10.1063/1.5089550.

[44] C. Ospelkaus et al. "Trapped-Ion Quantum Logic Gates Based on Oscillating Magnetic Fields". In: *Phys. Rev. Lett.* 101 (9 Aug. 2008), p. 090502. DOI: 10.1103/PhysRevLett.101.090502. URL: https://link.aps.org/doi/10.1103/PhysRevLett.101.090502.

[45] Robin Harper, Steven T. Flammia, and Joel J. Wallman. "Efficient learning of quantum noise". In: *Nature Physics* (2020). ISSN: 17452481. DOI: 10.1038/s41567-020-0992-8. arXiv: 1907.13022.

[46] Frank Arute et al. "Quantum supremacy using a programmable superconducting processor". In: *Nature* 574.7779 (2019), pp. 505–510.

[47] Yongshan Ding et al. "Systematic Crosstalk Mitigation for Superconducting Qubits via Frequency-Aware Compilation". In: (2020). arXiv: 2008.09503. URL: http://arxiv.org/abs/2008.09503.

[48] Prakash Murali et al. “Software mitigation of crosstalk on noisy intermediate-scale quantum computers”. In: *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*. Association for Computing Machinery, Mar. 2020, pp. 1001–1016. ISBN: 9781450371025. DOI: 10.1145/3373376.3378477. arXiv: 2001.02826.

[49] 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.

[50] Easwar Magesan et al. “Efficient measurement of quantum gate error by interleaved randomized benchmarking”. In: *Physical Review Letters* 109.8 (2012), pp. 1–5. ISSN: 00319007. DOI: 10.1103/PhysRevLett.109.080505. arXiv: 1203.4550.

[51] Timothy J. Proctor et al. “Direct Randomized Benchmarking for Multiqubit Devices”. In: *Physical Review Letters* 123.3 (2019), pp. 1–7. ISSN: 10797114. DOI: 10.1103/PhysRevLett.123.030503. arXiv: 1807.07975.

[52] David C. McKay et al. “Correlated Randomized Benchmarking”. In: (2020). arXiv: 2003.02354. URL: http://arxiv.org/abs/2003.02354.

[53] IBM. *qiskit.circuit.Delay*. https://qiskit.org/documentation/stubs/qiskit.circuit.Delay.html.

[54] IBM. *qiskit.transpiler.passes.NoiseAdaptiveLayout*. https://qiskit.org/documentation/stubs/qiskit.transpiler.passes.NoiseAdaptiveLayout.html.

[55] IBM. *qiskit.scheduler.schedule$_c$ircuit*. https://qiskit.org/documentation/stubs/qiskit.scheduler.schedule_circuit.html.

[56] Ang Li and Sriram Krishnamoorthy. *QASMBench: A Low-level QASM Benchmark Suite for NISQ Evaluation and Simulation*. 2020. arXiv: 2005.13018 [quant-ph].

[57] IBM. *qiskit.transpiler.passes.SabreLayout*. https://qiskit.org/documentation/stubs/qiskit.transpiler.passes.SabreLayout.html.

[58] IBM. *qiskit.compiler.transpiler*. https://qiskit.org/documentation/apidoc/transpiler.html.