IEEE Transactions on Quantum Engineering

Received January 18, 2022; revised March 7, 2022; accepted March 21, 2022; date of publication April 5, 2022; date of current version June 3, 2022.

Digital Object Identifier 10.1109/TQE.2022.3164716

Simultaneous Execution of Quantum Circuits on Current and Near-Future NISQ Systems

YASUHIRO OHKURA(大倉 康寛)[1,2], TAKAHIKO SATOH(佐藤 貴彦)[1,3], AND RODNEY VAN METER[1,4] (Senior Member, IEEE)

[1]Keio University Quantum Computing Center, Yokohama 223-8522, Japan
[2]Graduate School of Media and Governance, Keio University SFC, Fujisawa 252-0882, Japan
[3]Graduate School of Science and Technology, Keio University Faculty of Science and Technology, Yokohama 223-8522, Japan
[4]Faculty of Environment and Information Studies, Keio University SFC, Fujisawa 252-0882, Japan

Corresponding author: Yasuhiro Ohkura (e-mail: rum@sfc.wide.ad.jp).

This work was supported by MEXT Quantum Leap Flagship Program under Grant JPMXS0118067285.

ABSTRACT In the noisy intermediate-scale quantum (NISQ) era, the idea of *quantum multiprogramming*, running multiple quantum circuits (QCs) simultaneously on the same hardware, helps to improve the throughput of quantum computation. However, the crosstalk, unwanted interference between qubits on NISQ processors, may cause performance degradation when using multiprogramming. To address this challenge, we introduce *palloq* (parallel allocation of QCs), a novel compilation protocol. Palloq improves the performance of quantum multiprogramming on NISQ processors, while paying attention to 1) the combination of QCs chosen for parallel execution and 2) the assignment of program qubit variables to physical qubits, to reduce unwanted interference among the active set of QCs. We also propose a software-based crosstalk detection protocol using a new combination of randomized benchmarking methods. Our method successfully characterizes the suitability of hardware for multiprogramming with relatively low detection costs. We found a tradeoff between the success rate and execution time of the multiprogramming. Our results will be of value when device throughput becomes a significant bottleneck. Until service providers have enough quantum processors available to more than meet demand, this approach will be attractive to the service providers and users who want to optimize job management and throughput of the processor.

INDEX TERMS Compiler, crosstalk, multiprogramming, noisy intermediate-scale quantum (NISQ), quantum computing.

I. INTRODUCTION

Current processors, called noisy intermediate-scale quantum (NISQ) [1], are not immune to noise, which causes a high error rate and greatly affects the reliability of the computation.

With the advent of cloud quantum computing systems, quantum computing has become familiar to researchers and developers around the world. The more users and tasks from various demands and backgrounds have increased, the more it is important to maximize the throughput of the NISQ processors. To operate efficiently, executing multiple quantum tasks concurrently can be one solution. However, this method is not trivial and involves fundamental challenges [2]–[5].

It is difficult to explain the whole range of errors of computation on NISQ processors by using only information from standard error characterization techniques, such as randomized benchmarking [6], [7].

Though quantum tomography estimates the statistical reliability of a quantum state, it requires large amounts of data exponential in the number of qubits for complete state tomography. To maximize the utilization and performance of an NISQ processor, we should take into account not only the standard model of isolated qubits but also the detection model for context-dependent errors, which requires a small cost [8].

In the case of superconducting qubit systems, the effect of crosstalk on the gate errors is a serious problem [9], [10]. When multiple quantum circuits (QC) are executed in parallel, as resource usage of the processor increases, unwanted interference may occur due to crosstalk noise between independent QCs, which may affect the calculation results.

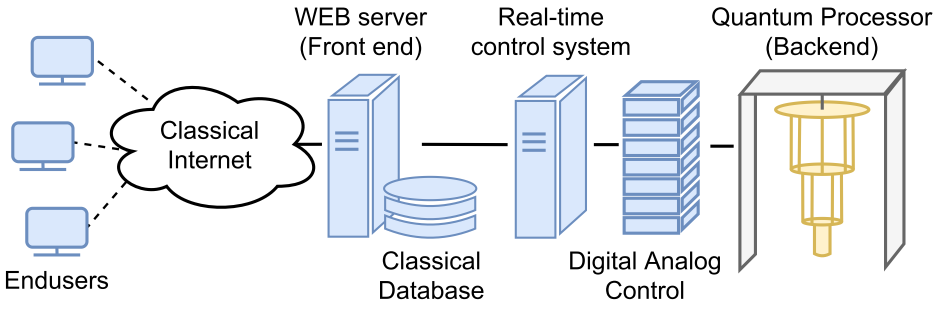


FIGURE 1. Cloud quantum computing architecture. Various implementations of quantum processors have been proposed [20]–[24]. As the core of the service, the quantum processor provides its computing resources by performing quantum operations and measurements. Digital analog control exchanges the quantum and classical information by converting program instruction into analog signals and measurement results into classical data. The real-time control system is responsible for classical-quantum interaction, mid-circuit measurements, and feed-forward operations. Another end of cloud computing is the browser-based user interface. End-users create the requests (jobs) on a web browser, send them to the system via a web server on the Internet, and receive the results of the computation. The browser-based user interface provides authentication, authorization, accountability (AAA), and in some cases, a quantum programming tool and its development environment, such as a GUI-based quantum circuit composer.

In this work, we introduce palloq (parallel allocation of QCs), a system for improving the performance of multiprogramming on NISQ processors. Our system consists of two parts: first, the multiple circuit composer to maximize searches for an effective combination of the application circuits (a knapsack-like problem); second, is the crosstalk-aware layout method that allocates hardware qubits to multiple QCs taking into account both local error rate and nonlocal noise (crosstalk). We also take into account the cost of crosstalk characterization and propose a novel detection method. We show the performance of our system by executing dozens to low hundreds of queued quantum tasks as a multiprogramming workload on the real-world cloud quantum computing platform, IBM quantum experience, and measure the success rate of the individual QCs and total execution time.

This article is organized as follows. In Section II, we review the cloud quantum computing environment, quantum multiprogramming, and crosstalk effect on NISQ processors. In Section III, first, we describe the crosstalk characterization approach in previous work, then introduce our novel characterization method. In Section IV, we propose compilation methods for efficient multiprogramming. In Section V, we show the experimental results on quantum processors and evaluate our proposed approach. Finally, Section VI concludes this article.

II. BACKGROUND

A. CLOUD QUANTUM COMPUTING

With the advent of cloud quantum systems, i.e., quantum computing as a service (QCaaS) [11]–[17], many researchers and developers from a variety of domains are becoming quantum users. QCaaS provides the quantum resource that allows opportunities ranging from conducting basic experiments [18] to developing applications that include quantum simulation, quantum machine learning, and optimization [19].

Cloud quantum computing architectures consist of components that include quantum processors, analog digital control, real-time control systems, WEB servers, classical databases, the Internet, and end-users, as in Fig. 1.

Why use a cloud quantum system instead of a local server or a quantum laptop? In general, there are several reasons for the migration of the service from desktop and corporate server rooms to a cloud platform [25], [26]. For the individual users, the total control of the software, OS, and low-level utility, and subsequent revisions to other programs comes with a price. For the service providers, the Internet-based service can be developed, tested, and operated on the platform provider's choice instead of coping with various user's environments. In the case of quantum computation, the development and operation, which includes daily calibration, of a quantum computer are very expensive and specialized tasks [27], [28]. Languages, tools, and environments for the development of the quantum program are still not sufficient. By providing them comprehensively as cloud services, users can utilize the quantum resources without being bothered by maintenance.

The rapid increase in users, urgent access to limited quantum resources, and the number of queued jobs are becoming serious issues.

B. PERFORMANCE OF QUANTUM COMPUTER

Several metrics for performance analysis of quantum computers have been proposed [29]–[31]. Quantum computing performance is governed by three factors: 1) the size of the problem that is encoded, which is determined by the number of physical qubits on a processor; 2) the size of a quantum circuit that can be faithfully executed, which is mainly determined by error rates of each operation and lifetime of qubits; and 3) the number of circuits that can be executed per unit time, which is related to the quantum and classical processing speed. In this article, we focus on the improvements of 2) the output fidelity of QCs, and 3) the number of QCs executed at a time by a quantum multiprogramming.

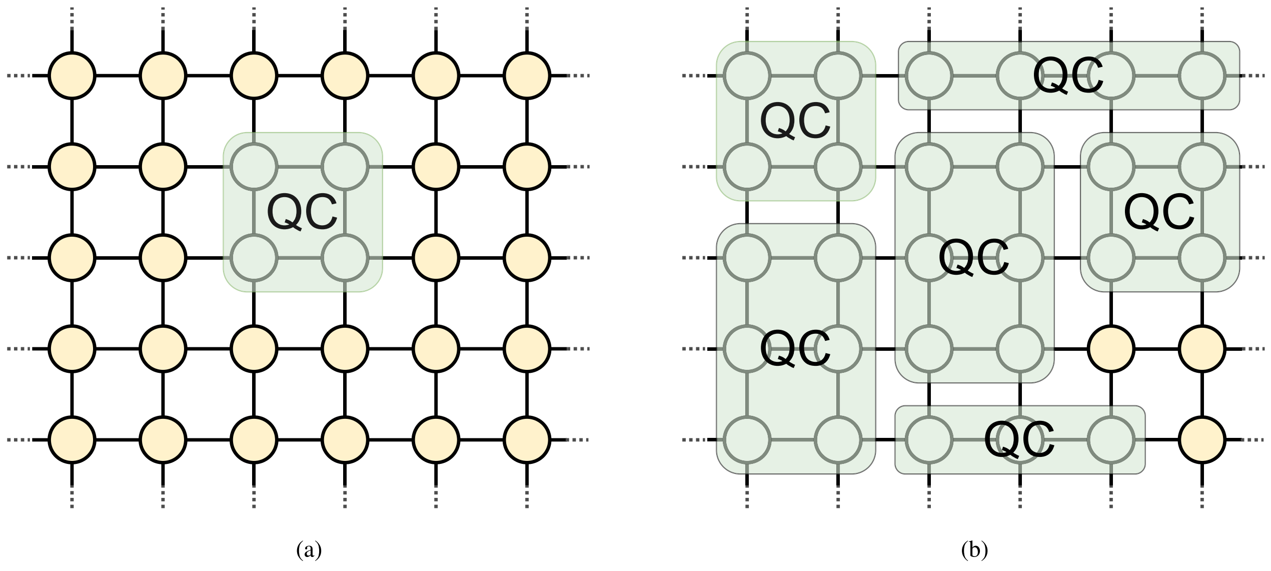


FIGURE 2. **Idea of multiple execution. The lattice graph represents the quantum processor, nodes denote qubits and edges are two qubits connection. The QC with the green box represents quantum circuits placed on physical qubits of the processor. (a) Single quantum circuit execution. Because the limited sized circuits are tolerable on NISQ system, the idle qubits (yellow circles) reduce the throughput of the processor. (b) Idea of multiple circuit execution concurrently. This approach reduces idle qubits and is expected to increase the throughput.**

C. QUANTUM MULTIPROGRAMMING

Quantum multiprogramming is a method for improving the throughput and utilization of the NISQ processor by executing multiple QCs simultaneously, instead of keeping the unemployed qubits idle, as shown in Fig. 2. In previous work, several challenges were discussed as follows [2], [3]:

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 including operational error rates and qubit lifetimes [27]. To solve this, the compiler needs to take this error information into account to optimize the circuit.
2) Avoidance of unwanted interference between the individual QC.
3) Optimization of the operational timing of each circuit to minimize the unnecessary decoherence effect.

In the case of multiprogramming several QCs with different depths (duration of execution), the shorter circuits suffer wait duration until the longer circuit's operation ends, which may cause the decaying of a quantum state prepared by shorter circuits and reduce the output reliability.

Improving the utilization of the processor by executing multiple programs concurrently can increase the unwanted interference between independent QCs. To reduce serious destructive interference, one option is to monitor and compare the performance of parallel execution and to feed the result into the next execution phase, either single or in parallel [2]. Rather than that, Ohkura [4], and Dou and Liu [5] discussed directly focusing on the crosstalk noise on the device which causes nonlocal errors on QCs of multiprogramming. They tried to characterize the crosstalk in the processor and optimize qubit allocation along with it. The problem is the crosstalk characterization grows quadratically in the number of hardware qubits, as we discuss in Section III.

D. CROSSTALK IN NISQ PROCESSOR

Crosstalk is known to be a significant source of noise in the quantum processor. This type of error can be explained from several aspects, but it is simply the unwanted interaction between coupled qubits in the processor. It is known that there is a tradeoff between the strength of qubit interaction and the magnitude of unwanted crosstalk noise [32], [33]. One type of crosstalk is caused by simultaneous operations between specific pairs of qubits. In this article, we focused on the unwanted interaction due to the two-qubit (CX gate) operations. This type of crosstalk is known to occur in the current quantum architectures including superconducting systems and trapped ions [34], [35].

The tuning and mitigation of crosstalk directly become big challenges when developing larger processors [8], [33], [36]. There are several software approaches to reducing crosstalk errors introduced in previous work. In the case of tunable quantum processors including Google's architecture [37], we can tune qubit frequencies or control specific couplers to disable and shut down the leakage errors [9], [38]. In contrast, in fixed frequency qubit systems including IBM Q system, we can optimize the circuit scheduler to avoid concurrent execution of correlated qubits in the processor [39]. In this article, we focused on this fixed qubit system, and provide the

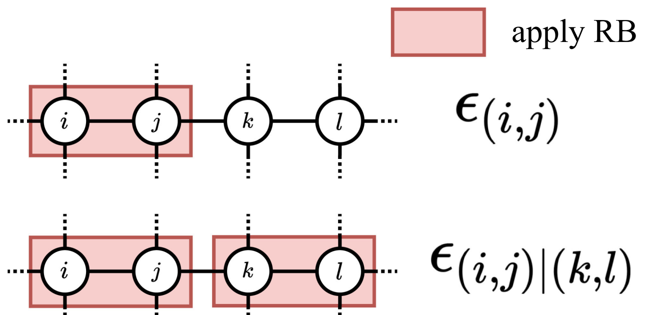


FIGURE 3. **Simultaneous randomized benchmarking. Upper diagram shows ordinary RB on two-qubits. RB applies random Clifford operations with varying the length of gates and estimates its error rates. In the case of Simultaneous RB, applying RB on more than two hardware areas and comparing the error rates to a single RB case, measure the conditional error rates of hardware qubits. In this research, we only conducted two qubits RB and SimRB.**

solution by a novel layout method in the circuit compilation process.

III. CROSSTALK CHARACTERIZATION

To understand the performance of quantum processors affected by local and nonlocal errors and their ability to concurrently execute multiple QCs, we need to characterize crosstalk in the processor. To simplify the problem, we only take into account how big the average crosstalk on the processor is rather than where these occur, i.e., the location of qubits.

A. COST OF CHARACTERIZATION

The complexity of crosstalk characterization often scales exponentially with the system size. Recently, several works showed ways to suppress the cost of crosstalk characterization. One practical protocol introduced in [5] and [39] is the comparison of the error rate in the case of individual and parallel execution by using simultaneous randomized benchmarking (SimRB). For example, in Fig. 3, some pairs of two-qubit errors can be detected in parallel, e.g., (q_i, q_j) and (q_k, q_l).

If the error rate from SimRB $\epsilon(q_i, q_j)|(q_k, q_l)$ is significantly different from the individual RB $\epsilon(q_i, q_j)$, there is an unwanted correlation between them.

The combination of these two-qubit pairs grows quadratically in the size of the processor. To avoid this situation, in a previous study [39], Murali *et al.* provide some rules to reduce this overhead. 1) Characterize one-hop pairs. Through the experiments, they found the tendency of occurrence of crosstalk is limited only at one hop on the IBM Q system they used. This rule is suppressing the detection cost by ignoring the pairs more than one hop apart. They also pointed out some older devices have long-range crosstalk strong enough to be a concern. 2) Characterize high crosstalk pairs only.

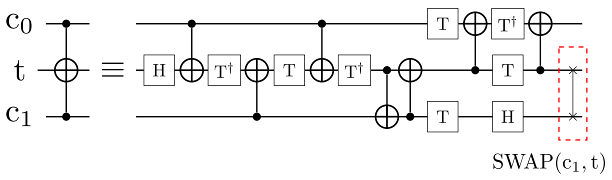


FIGURE 4. **Toffoli gate on chain topology.**

They also found the existence of crosstalk tends to be stable in time and space.

Although this method can detect crosstalk distribution on the processors, it still takes several hours, and it may be impractical in the current situation because the size of processors is continuously getting bigger, and these experiments are queued and run on the cloud system in the presence of other participants.

B. PHYSICAL BUFFER AND SUCCESS RATE

In this section, we introduce a physical buffer, which is the number of idle qubits between QCs, to mitigate the crosstalk effect of the concurrent execution of multiple QCs. We conducted a preliminary experiment to quantify how the physical buffer affects the output reliability of individual tasks in the concurrent execution. We used the Toffoli gate as the benchmark. Due to the compilation, which includes the SWAP gate that consists of three CX gates and the qubit routing along with the topology of the processor, shown in Fig. 4, for a total of 10 CX gates.

We varied the number of physical buffers between circuits and the number of Toffoli gates as parameters to see how the success rates change, as shown in Fig. 5. Fig. 6 shows that in only the case of adjacent circuits, i.e., no physical buffer, the success rate significantly drops. With the increase in the number of buffers, the success rate is recovered. And this change appears only for circuits of more than 20 CX gates. This leads us to the following insights. For the concurrent execution in practice, 1) we can ignore the crosstalk for the shorter circuit and 2) there is a threshold number of hops of the physical buffers that can improve the output fidelity. And in this specific case, execution of multiple Toffoli ranging the CX gates depth 10 to 30 on the processor IBMQ Manhattan, 1 physical buffer is enough until the CX circuit depth reaches 30. We found this second threshold varies depending on the processors that run multiple circuits and what size of QCs are executed in a later experiment is shown in Figs. 8 and 10.

C. BENCHMARK OF CROSSTALK IMMUNITY

Although crosstalk is the major source of error and may decrease the reliability of the concurrent execution, with the increasing number of qubits in the processors, the detection cost of running a set of quantum operations and measurements for characterizing the noise, increases quadratically.

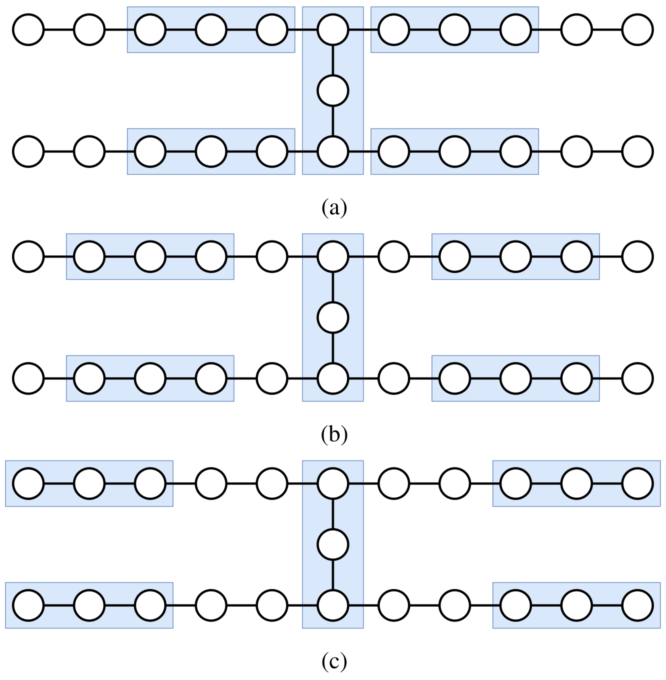


FIGURE 5. Multiple Toffoli placement varying physical buffer. The graph denotes the quantum processor, nodes are qubits and edges are the two-qubit connection of the superconducting qubit system. The blue boxes represent the quantum circuits of the Toffoli operation placed on physical qubits. We placed five circuits and vary the physical buffer, then measure the success rate of the Toffoli placed in the center. (a) No physical buffers. (b) One physical buffer. (c) Two physical buffers.

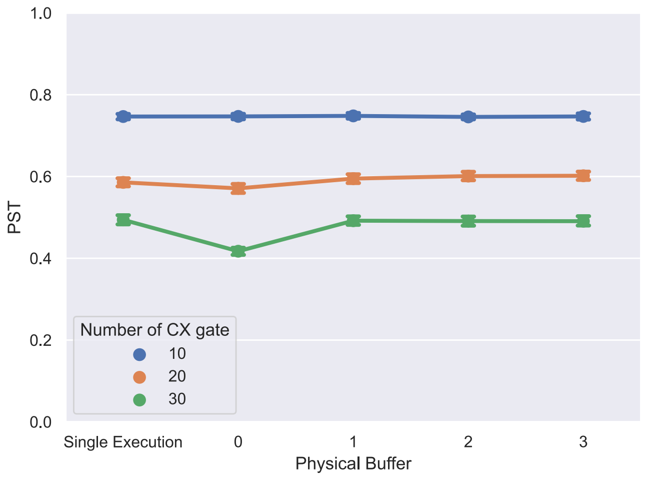


FIGURE 6. Physical buffer of multiple circuits and success rate. Each line denotes probability of successful trial (1) of Toffoli gates in the case of concurrent execution. The horizontal axis is the physical qubit buffer introduced in Fig. 5. We compared the PST to the left most data point, which is the single execution case. Each color describes the number of CX gates contained in the operation, 10, 20, and 30. Since we utilize the Toffoli gates with SWAP Fig. 4, one toffoli gate contains 10 CX gates.

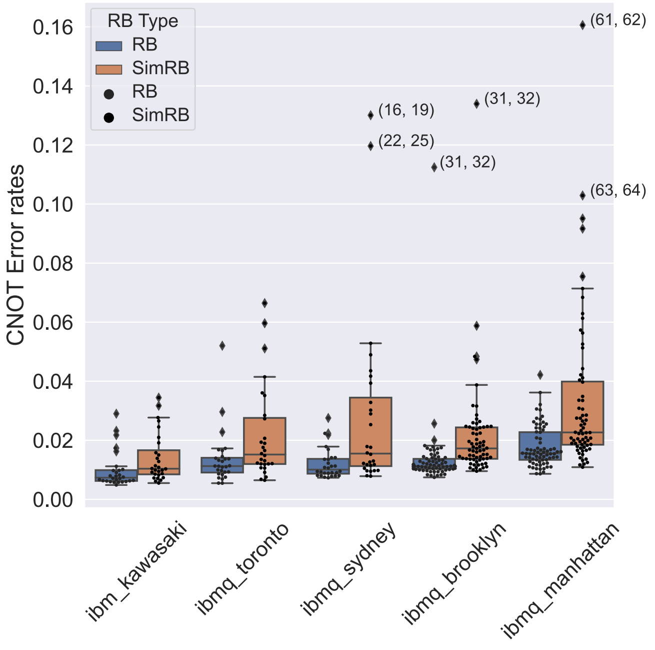


FIGURE 7. Crosstalk immunity Each black dot represents CX gate error rate of two qubit pairs. (i, j) denotes the label of two-qubit pair which is remarkably high error rate.

We show a software-based crosstalk detection protocol using a new combination of simultaneous randomized benchmarking techniques with relatively low detection costs. To analyze the crosstalk tolerance of processors, we utilized the coefficient of variation of gate errors as the metric and compared several processors.

We focus on how the crosstalk impacts the average and variance of error rates. First, we apply RB for every qubit or two-qubit pair in the processor and calculate the average and variance of error rates in the single execution case. Then, we run SimRB for all the qubits at the same time and also calculate the average and variance. Comparison of average and variance of error rates between those two cases leads to the quantitative analysis of crosstalk effects on the whole performance and immunity of the processor. We conducted this benchmark on several current IBM quantum processors and showed the comparison of those performances in Fig. 7. We measured the CX gate error rate of each processor by utilizing RB. Blue box plots represent the distribution of the CX gate error rate and black dots are the error value of each physical two-qubit connection. The orange box plots also represent error rates but in the case of concurrent execution (SimRB). We ran several patterns of CX gates combination that can be executed concurrently and took the average value of error rates of each case.

For all processors, the variance and average error rates increase in the case of concurrent execution, indicating the presence of the crosstalk noise. In particular, IBMQ Toronto, IBMQ Sydney, and IBMQ Manhattan show significant interference by other operations performed in the other regions. The performance of multiprogramming directly depends on this crosstalk interference as we discuss in Section V.

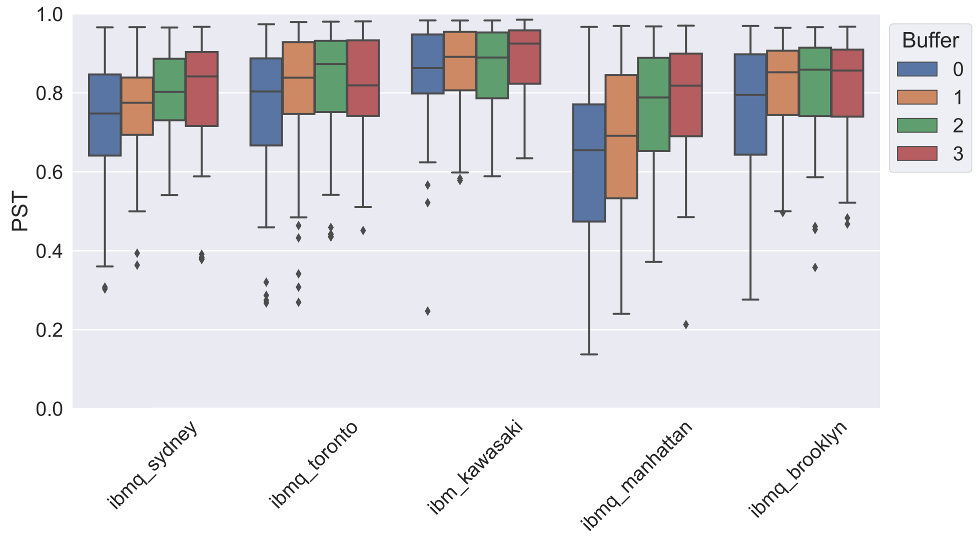


FIGURE 8. **Success rate and the physical buffer. These box plots represent the distribution of PST of benchmark circuits executed on the real quantum devices.**

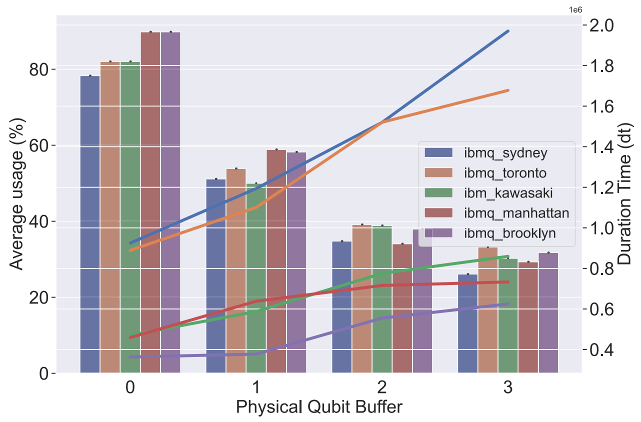


FIGURE 9. **Hardware usage and total execution time.**

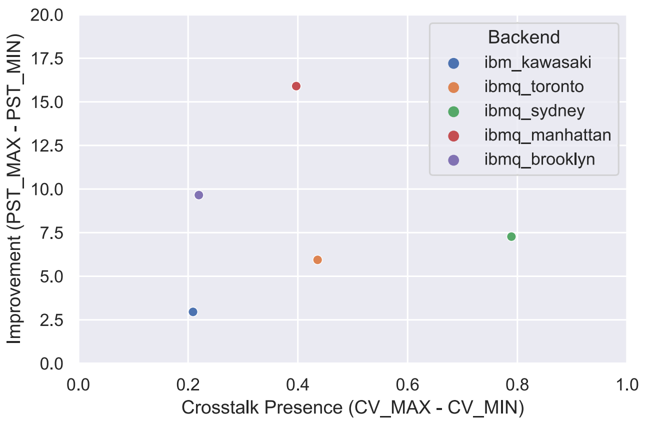


FIGURE 10. **Crosstalk presence and improvement of concurrent execution.**

IV. PALLOQ SYSTEM

We propose *palloq*, a system including layout synthesis for multiple QCs and a job scheduler to manage efficient and high fidelity quantum multiprogramming. The detail of the procedure is explained in Appendix B. We published the source code at https://github.com/rum-yasuhiro/palloq.

The palloq is responsible for the compilation phase and the user's job management, which is provided by the WEB server in the cloud computing architecture we showed in Fig. 1.

This compiler pass takes several QCs written in OpenQASM [40] and the local gate error information of the device as input.

Our layout synthesis consists of a heuristic based on noise-adaptive layout, which analyzes the device's calibration data and searches for better allocation using a greedy approach [41]. First, it parses the calibration data and hardware qubit connection to create a weighted graph $G_{HW}(V, E)$. V, E, and weight represent the physical qubits as the vertices, two-qubit connections of the hardware as the edges, and the reliability $r = 1 - \epsilon$, where ϵ is the error rate of two-qubit operation between physical qubits. In the same way, each input QC is treated as a weighted directed graph $G_{QC}(V, E)$, where vertices V are the qubits in the QC, edges E are two-qubit gates, and weight is the number of two-qubit gates performed on the same two-qubit pair. The compiler searches for the best reliable physical qubits candidates heuristically and allocates them to the highest weighted edges in the circuit's graph. Repeat this procedure until all the circuit components are placed.

As we discussed in Section III-B, for the shorter circuits, we do not care about the crosstalk, and for the relatively larger circuits, we only care about the physical distance

TABLE 1. **Small Benchmark Circuits**

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

We picked several small-size quantum circuits to benchmark our proposal from the benchmark circuit set called QASMBench [42].

TABLE 2. **Processors**

Name	Spec
Intel Core i9 processor	2.3 GHz, 32 GB RAM
IBMQ Toronro	Falcon r4 27-qubit QV32
IBMQ Sydney	Falcon r4 27-qubit QV32
IBM Kawasaki	Falcon r5.11 27-qubit QV32
IBMQ Manhattan	Hummingbird r2 65-qubit QV32
IBMQ Brooklyn	Hummingbird r2 65-qubit QV32

TABLE 3. **Software Version**

Name	Version
Python	3.8.5
qiskit	0.29.0
qiskit-terra	0.17.0
qiskit-aer	0.8.2
qiskit-ignis	0.5.1
qiskit-ibmq-provider	0.16.0

between circuits and optimize them locally. Our software takes physical qubit distance as input. Every time each circuit is allocated to hardware qubits, then disable the qubits around them to create a distance to others.

V. EXPERIMENTS AND EVALUATION

To evaluate our proposal, we conducted an experiment varying the physical buffer among the multiple circuits. In the entire experiment, we focus on the output reliability and total execution time, and hardware usage.

We use small benchmark circuits from previous work [42], as shown in Table 1. The details of the processors and the software we used are shown in Tables 2 and 3.

A. METRICS

To quantify the performance of our proposed method, we utilized probability of successful trial (PST) [27]. For NISQ systems that run the given programs a specified number of times, known as "shots," the quantum computation outputs the answer as a probability distribution of candidate bit strings. The number of successful trials is defined as how many times each shot hit the correct bit strings. We chose the benchmark circuits whose computational result includes only one or a small number of answers for the solution space for ease of handling with PST. When the QC finishes successfully without any error, PST is 1.

As a baseline, we compare the results from a noisy quantum device to a classical simulator as a noiseless case. PST is defined as follows:

$$\text{PST} = \frac{\text{Number of Successful Trials}}{\text{Number of Total Trials}}. \tag{1}$$

B. EXECUTION DURATION AND OUTPUT RELIABILITY

First, we prepared 100 QCs from the benchmark set and queued them as input to our compiler. Varying the physical qubit buffer 0 to 3, we count the output PST and total execution time of concurrent execution.

Fig. 8 shows that PST of 100 circuits varies with physical buffer. For all quantum devices, in the case of buffer more than one, the average PST is better than the case of buffer zero, which is the densest layout and highest throughput case. This means, in these experiments, we obtain higher reliability of computation at the expense of throughput of concurrent execution. Compared to the result from Fig. 6, increasing the buffer distance results in continuous improvement in the PST in this experiment. This difference is due to differences in the quantum circuits. In the experiment shown in Fig. 6, all circuits are of a fixed Toffoli gate depth. In this experiment, once benchmark circuits are assigned to physical qubits on the processor, a routing pass is applied in the compilation process that transforms the circuit structure. This adds gate overhead even as we optimize for the processor's physical qubit coupling graph. Depending on the types of QCs, the increase in gate depth causes a stronger crosstalk effect. Though IBMQ Brooklyn is the same architecture as IBMQ Manhattan, base PSTs are higher than IBMQ Manhattan, and improvement stops when we use more than two physical buffers. This difference comes from the presence of crosstalk we found in Fig. 7.

Fig. 9 shows the total execution time of 100 circuits as we vary the physical buffer. For the execution time, we count circuit duration time (dt) of all concurrent execution round and 1 $dt = \frac{2}{9}$ ns. It shows a larger buffer reduces hardware usage and total execution time linearly. For all devices, around 80% of physical qubits are used in the case of buffer 0. In the case of buffers 2 and 3, the hardware usage is only half of the densest case. On the other hand, for all devices, the total circuit duration time increased by a factor of two from the densest to the sparsest.

TABLE 4. Date and Time When Experimental Data Have Been Taken

Experiment	Date-time
Physical buffer of multiple circuits in Fig. 6 on IBMQ Manhattan	2021/05/10 2021/05/14
Crosstalk detection in Fig. 7 on IBMQ Toronto	2021/07/09
Crosstalk detection in Fig. 7 on IBMQ Sydney	2021/07/09
Crosstalk detection in Fig. 7 on IBM Kawasaki	2021/07/09
Crosstalk detection in Fig. 7 on IBMQ Manhattan	2021/07/10
Crosstalk detection in Fig. 7 on IBMQ Brooklyn	2021/07/10
Performance analysis of proposal compiler in Fig. 8 on IBMQ Toronto	2021/08/25
Performance analysis of proposal compiler in Fig. 8 on IBMQ Sydney	2021/08/25
Performance analysis of proposal compiler in Fig. 8 on IBM Kawasaki	2021/08/25
Performance analysis of proposal compiler in Fig. 8 on IBMQ Manhattan	2021/08/25
Performance analysis of proposal compiler in Fig. 8 on IBMQ Brooklyn	2021/08/25

C. ANALYSIS

Here we describe the tradeoff between crosstalk and throughput of concurrent execution. In general, we desire the success of computation with higher reliability and throughput. The success rate of concurrent computation depends on the gate fidelity (local errors) and crosstalk. The throughput, which is defined as the number of executed QC per dt, where dt is the duration time of quantum computation, relies on many factors like gate duration time, measurement duration time, and the size of processors. Based on experimental results shown in Fig. 8, we defined the improvement (gain) of output reliability g as

$$g = \max(\mathrm{PST_i} - \mathrm{PST_j}) \tag{2}$$

where the PST_i is the average value of PSTs of i physical qubit buffered layout of concurrent execution. In this experiment, for all cases, $i = 0$ and j is 2 or 3. Based on the result shown in Fig. 7, we defined the crosstalk presence in the processor ct as

$$ct = CV_{\mathrm{SimRB}} - CV_{\mathrm{RB}} \tag{3}$$

utilizing the coefficient of variation (CV)

$$CV = \frac{\sigma}{\mu} \tag{4}$$

where σ is the standard deviation and μ is the average of the given distribution. In this case, crosstalk presence ct represents the degree of change of variation of CX error rate from RB to SimRB.

Fig. 10 shows the relationship between the presence of crosstalk on the device and gain of improvement by the number of physical qubit buffers.

We used three 27-qubit devices and two 65-qubit devices, and in both cases, we can see a positive correlation. Referring to Fig. 9, as we increase the number of physical qubit buffers, the throughput of computation drops. For processors with relatively strong crosstalk, it is worth reducing the throughput to gain the improved output fidelity of the computation.

VI. CONCLUSION

This article proposed and evaluated a compiler method for concurrent execution of multiple quantum circuits, which includes layout and schedule. First, we showed a practical crosstalk characterization method to reduce the detection cost that is critical for the near-term scale quantum processors. For the evaluation, we show our compiler efficiently processes the multiple quantum circuits avoiding the crosstalk and tradeoff between the success rate of quantum circuits and the throughput of the processors.

As a further research direction, there are two possible paths. In this article, we deal with a static size queue of quantum circuits. In a realistic situation, the optimization is more complicated due to the dynamic queue varying in time. Also, we only consider only one quantum processor at a time. If we can utilize several processors to deal with multi-programming, there is plenty of room to optimize. In the other direction, the optimization approach might be improved by a more advanced solution to the knapsack problem, such as using dynamic programming (DP). Also, we mentioned the limitations of our approach in Section V-B, that some combination of allocation of the quantum circuits leads to higher overhead in circuit depth, causing serious crosstalk. One potential solution is finding the initial allocation of multiple quantum circuits (e.g., via simulated annealing) with the cost function set to be the depth of the compiled quantum circuit.

APPENDIX

A. SETUP FOR EXPERIMENTS

The quantum circuit for benchmarking our proposed software is listed in Table 1. The processors we use are listed in Table 2 and the version of software packages we use are listed in Table 3.

Each experiment was conducted on the dates listed in Table A4.

B. PSEUDOCODE

Algorithm 1 shows the pseudocode we introduced in Section IV.

Algorithm 1: Physical Distance Layout.

```
Input: Queued DAG circuits $queue$, Processor's topology graph weighted by CX gate reliability $G_{HW}$, Buffer to
    adjacent circuit $d$
Output: List of Multi-programming circuits with layout as MP

Set MP to empty list;
$queue = sort\{queue \mid CX\ depth\}$;
$S := \{G_{HW_S} \in G_{HW} \mid connected\ subgraph\ of\ G_{HW}\}$;
while $length(queue) > 0$ do
    $DAG = queue.pop(0)$;
    if $size(max(S)) > size(DAG)$ then
        Set layout, pending_qubits, prog, hw to empty list;
        Set multi_QC to empty DAG;
        $G_{QC} \xleftarrow{convert} DAG$;   /* Convert DAG to graph $G_{QC}$ weighted by number of CX gates
          */
        $pending\_qubits \leftarrow G_{QC}.nodes$;
        $q_{prog1}, q_{prog2} \leftarrow max\{G_{QC}.edges\}$;                /* Heaviest program edge */
        $pending\_qubits.remove(q_{prog1}, q_{prog2})$
        $prog.append(q_{prog1}, q_{prog2})$;
        $q_{hw1}, q_{hw2} \leftarrow max\{G_{HW_S}.edges \mid size(G_{HW_S}) > size(DAG)\}$; /* Most reliable HW qubits */
        while $pending\_qubits\ not\ null\ set$ do
            $q_{prog} \leftarrow q_{adj} \in max\{edge(q_{adj}, q_{in}) \in G_{QC} \mid q_{in} \in prog\}$;
            $q_{hw} \leftarrow q_{adj} \in max\{edge(q_{adj}, q_{in}) \in G_{HW_S} \mid q_{in} \in hw\}$;

            $pending\_qubits.remove(q_{prog})$

            $layout.append(\{q_{prog}, q_{hw}\})$;
            $prog.append(q_{prog})$;
            $hw.append(q_{hw})$;
        end while
        Delete used qubits;
        $G_{HW}.remove(hw)$;
        $G_{HW}.remove(q \in G_{HW} \mid d - neighbored\ to\ hw)$;
        $S := \{G_{HW_S} \in G_{HW} \mid connected\ subgraph\ of\ G_{HW}\}$;
        $multi\_QC.append(DAG)$
    else
        $MP.append(\{multi\_QC, layout\})$;
        Reinitialize hardware graph $G_{HW}$;
        $S := \{G_{HW_S} \in G_{HW} \mid connected\ subgraph\ of\ G_{HW}\}$;
    end if
end while
return MP
```

REFERENCES

[1] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, 2018, Art. no. 79, doi: 10.22331/q-2018-08-06-79.

[2] P. Das *et al.*, "A case for multi-programming quantum computers," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2019, pp. 291–303, doi: 10.1145/3352460.3358287.

[3] X. Dou and L. Liu, "A new qubits mapping mechanism for multi-programming quantum computing," in *Proc. ACM Int. Conf. Parallel Archit. Compilation Techn.*, 2020, pp. 349–350, doi: 10.1145/3410463.3414659.

[4] Y. Ohkura, "Crosstalk-aware NISQ multi-programming," Faculty Policy Manage., Keio Univ., Tokyo, Japan, 2021.

[5] S. Niu and A. Todri-Sanial, "Enabling multi-programming mechanism for quantum computing in the NISQ era," 2021, *arXiv:2102.05321*, doi: 10.48550/arXiv.2102.05321.

[6] E. Knill *et al.*, "Randomized benchmarking of quantum gates," *Phys. Rev. A.*, vol. 77, 2008, Art. no. 012307, doi: 10.1103/PhysRevA.77.012307.

[7] J. M. Chow *et al.*, "Randomized benchmarking and process tomography for gate errors in a solid-state qubit," *Phys. Rev. Lett.*, vol. 102, 2009, Art. no. 090502, doi: 10.1103/PhysRevLett.102.090502.

[8] K. Rudinger *et al.*, "Probing context-dependent errors in quantum processors," *Phys. Rev. X*, vol. 9, no. 2, 2019, Art. no. 021045, doi: 10.1103/PhysRevX.9.021045.

[9] P. Mundada *et al.*, "Suppression of qubit crosstalk in a tunable coupling superconducting circuit," *Phys. Rev. Appl.*, vol. 12, no. 5, 2019, Art. no. 054023, doi: 10.1103/PhysRevApplied.12.054023.

[10] D. C. McKay *et al.*, "Three-qubit randomized benchmarking," *Phys. Rev. Lett.*, vol. 122, no. 20, 2019, Art. no. 200502, doi: 10.1103/PhysRevLett.122.200502.

[11] Amazon, Seattle, WA, USA, "Amazon braket," [Online]. Available: https://aws.amazon.com/braket/

[12] D-wave, Burnaby, BC, Canada, "D-wave leap," https://cloud.dwavesys.com/leap/

[13] Google, Mountain View, CA, USA, "Quantum computing playground," [Online]. Available: https://experiments.withgoogle.com/quantum-computing-playground

[14] IBM, Armonk, NY, USA, "IBM quantum experience," [Online]. Available: https://quantum-computing.ibm.com/

[15] Microsoft, Albuquerque, NM, USA, "Microsoft azure quantum," [Online]. Available: https://azure.microsoft.com/en-us/services/quantum/

[16] Rigetti, Berkeley, CA, USA, "Rigetti forest," [Online]. Available: https://www.rigetti.com
[17] Xanadu, Toronto, ON, Canada, "Xanadu qauntum cloud, [Online]. Available: https://www.xanadu.ai
[18] S. J. Devitt, "Performing quantum computing experiments in the cloud," *Phys. Rev. A*, vol. 94, no. 3, 2016, Art. no. 032329, doi: 10.1103/PhysRevA.94.032329.
[19] A. Montanaro, "Quantum algorithms: An overview," *NPJ Quantum Inf.*, vol. 2, no. 1, 2016, pp. 1–8, doi: 10.1038/npjqi.2015.23.
[20] J. A. Jones, "Quantum computing and nuclear magnetic resonance," *Phys. Chem. Commun.*, vol. 4, no. 11, 2001, pp. 49–56, doi: 10.1039/B103231N.
[21] J. Clarke and F. K. Wilhelm, "Superconducting quantum bits," *Nature*, vol. 453, no. 7198, 2008, pp. 1031–1042, doi: 10.1038/nature07128.
[22] D. Kielpinski, C. Monroe, and D. J. Wineland, "Architecture for a large-scale ion-trap quantum computer," *Nature*, vol. 417, no. 6890, 2002, pp. 709–711, doi: 10.1038/nature00784.
[23] P. Hemmer and M. Lukin, "Quantum register based on individual electronic and nuclear spin qubits in diamond," *Science*, vol. 316, no. 5829, 2007, pp. 1312–1316, doi: 10.1126/science.1139831.
[24] J. L. O'brien, A. Furusawa, and J. Vučković, "Photonic quantum technologies," *Nature Photon.*, vol. 3, no. 12, 2009, pp. 687–695, doi: 10.1038/nphoton.2009.229.
[25] B. Hayes, "Cloud computing," *Commun. ACM*, vol. 51, no. 7, pp. 9–11, Jul. 2008, doi: 10.1145/1364782.1364786.
[26] M. Armbrust *et al.*, "A view of cloud computing," *Commun. ACM*, vol., vol. 53, no. 4, pp. 50–58, Apr. 2010, doi: 10.1145/1721654.1721672.
[27] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: A case for variability-aware policies for NISQ-era quantum computers," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Oper. Syst.*, 2019, pp. 987–999, doi: 10.1145/3297858.3304007.
[28] D. C. McKay *et al.*, "Qiskit backend specifications for OpenQASM and experiments," 2018, *arXiv:1809.03452*, doi: 10.48550/arXiv.1809.03452.
[29] A. D. Córcoles *et al.*, "Exploiting dynamic quantum circuits in a quantum algorithm with superconducting qubits," *Phys. Rev. Lett.*, vol. 127, no. 10, 2021, Art. no. 100501, doi: 10.48550/arXiv.1809.03452.
[30] T. Lubinski *et al.*, "Application-oriented performance benchmarks for quantum computing," 2021, *arXiv:2110.03137*, doi: 10.48550/arXiv.2110.03137.
[31] A. Wack *et al.*, "Quality, speed, and scale: Three key attributes to measure the performance of near-term quantum computers," 2021, *arXiv:2110.14108*, doi: 10.48550/arXiv.2110.14108.
[32] J. M. Gambetta *et al.*, "Characterization of addressability by simultaneous randomized benchmarking," *Phys. Rev. Lett.*, vol. 109, no. 24, 2012, Art. no. 240504, doi: 10.1103/PhysRevLett.109.240504.
[33] S. Sheldon *et al.*, "Procedure for systematically tuning up cross-talk in the cross-resonance gate," *Phys. Rev. A.*, 93, no. 6, 2016, Art. no. 060302, doi: 10.1103/PhysRevA.93.060302.
[34] P. Krantz *et al.*, "A quantum engineer's guide to superconducting qubits," *Appl. Phys. Rev.*, vol. 6, no. 2, 2019, Art. no. 021318, doi: 10.1063/1.5089550.
[35] C. Ospelkaus *et al.*, "Trapped-ion quantum logic gates based on oscillating magnetic fields," *Phys. Rev. Lett.*, vol. 101, no. 9, 2008, Art. no. 090502, doi: 10.1103/PhysRevLett.101.090502.
[36] R. Harper, S. T. Flammia, and J. J. Wallman, "Efficient learning of quantum noise," *Nature Phys.*, vol. 16, no. 12, 2020, pp. 1184–1188, doi: 10.1038/s41567-020-0992-8.
[37] F. Arute *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, pp. 505–510, 2019, doi: 10.1038/s41586-019-1666-5.
[38] Y. Ding, P. Gokhale, S. F. Lin, R. Rines, T. Propson, and F. T. Chong, "Systematic crosstalk mitigation for superconducting qubits via frequency-aware compilation," *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2020, pp. 201–214, doi: 10.1109/MICRO50266.2020.00028.
[39] P. Murali, D. C. McKay, M. Martonosi, A. Propson, and F. T. Javadi-Abhari, "Software mitigation of crosstalk on noisy intermediate-scale quantum computers," *Proc. 25th Int. Conf. Architectural Support Program. Lang. Oper. Syst.*, 2020, pp. 1001–1016, doi: 10.1145/3373376.3378477.
[40] A. W. Cross *et al.*, "Open quantum assembly language," 2017, *arXiv:1707.03429*, doi: 10.48550/arXiv.1707.03429.
[41] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonos, "Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers," *Proc. 24th Int. Conf. Architectural Support Program. Lang. Oper. Syst.*, 2019, pp. 1001–1016, doi: 10.1145/3297858.3304075.
[42] A. Li *et al.*, "QASMbench: A low-level QASM benchmark suite for NISQ evaluation and simulation," 2020, doi: 10.48550/arXiv.2005.13018.

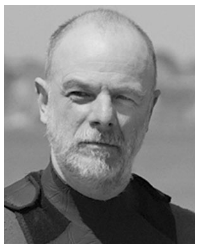
Yasuhiro Ohkura received the B.S. degree in policy management, in 2021, from Keio University, Tokyo, Japan, where he is currently working toward the M.S. degree in media and governance.

His research interests include the development of the NISQ algorithm and post-NISQ computing, the design of the compiler for quantum computation, quantum error mitigation, and quantum computer architecture.

Takahiko Satoh received the Ph.D. degree in computer science from the University of Tokyo, Tokyo, Japan, in 2016.

He is a project Assistant Professor with Keio University Quantum Computing Center, Keio University, Tokyo, Japan. His research interests include quantum computing and quantum networking, particularly quantum network coding, NISQ algorithm design, and quantum Internet security.

Dr. Satoh is a member of the Physical Society of Japan.



Rodney Van Meter (Senior Member, IEEE) received the Ph.D. degree in computer science from Keio University, Tokyo, Japan.

He is a Professor of Environment and Information Studies, Keio University, Shonan Fujisawa Campus, Japan. His research interests include storage systems, networking, and post-Moore's law computer architecture, besides quantum networking and quantum computing.

Dr. van Meter is a Member of ACM, the American Physical Society, and the American Association for the Advancement of Science.