Systematic Crosstalk Mitigation for Superconducting Qubits via Frequency-Aware Compilation

Yongshan Ding, Pranav Gokhale, Sophia Fuhui Lin
Richard Rines, Thomas Propson, and Frederic T. Chong

Department of Computer Science, University of Chicago, Chicago, IL 60615, USA

***Abstract*—One of the key challenges in current Noisy Intermediate-Scale Quantum (NISQ) computers is to control a quantum system with high-fidelity quantum gates. There are many reasons a quantum gate can go wrong – for superconducting transmon qubits in particular, one major source of gate error is the unwanted crosstalk between neighboring qubits due to a phenomenon called frequency crowding. We motivate a systematic approach for understanding and mitigating the crosstalk noise when executing near-term quantum programs on superconducting NISQ computers. We present a general software solution to alleviate frequency crowding by systematically tuning qubit frequencies according to input programs, trading parallelism for higher gate fidelity when necessary. The net result is that our work dramatically improves the crosstalk resilience of tunable-qubit, fixed-coupler hardware, matching or surpassing other more complex architectural designs such as tunable-coupler systems. On NISQ benchmarks, we improve worst-case program success rate by 13.3x on average, compared to existing traditional serialization strategies.**

***Index Terms*—quantum computing, error mitigation, compiler optimization, superconducting qubit**

I. Introduction

Current Noisy Intermediate-Scale Quantum (NISQ) computers [2], [26], [27], [43], [52] aim to isolate and control a non-trivial quantity of quantum bits (qubits) with high precision. Scaling up a quantum computer requires improvements in both the quality of qubits (with longer lifetime) and the quality of gates (with higher fidelity).

In case of superconducting transmon qubits [4], [25], [32], which is the subject of this work, gate speeds have been achieved three to four orders of magnitude faster than qubit lifetime [3], [6], [12], [45]. Although fast gates are desirable; they are prone to errors caused by imprecise control. Among all sources of gate errors, crosstalk is the most dominant [37], [38]. Errors caused by crosstalk, such as exchange of excitation and leakage to non-computational states, are found to have detrimental effect to quantum states, and such errors can accumulate as we execute a program [3].

What is crosstalk? There is hardly a single precise noise model that captures all aspects of crosstalk, but rather, it is a combination of *unwanted interactions between coupled qubits* on a quantum chip. This type of crosstalk noise prevails in many leading architectures, including trapped ion and superconducting systems [13], [33], [41], [42]. For superconducting transmon systems, two qubits interact with each other via *resonance of qubit frequency*. Two main technology options for avoiding accidental resonance of qubits are: i) to tune qubit frequencies apart using tunable qubits; ii) to temporarily disable connections between qubits using tunable couplers. Fig. 1 illustrate the different design choices of leading QC architectures. Current IBM Q systems [26] are built with fixed qubit frequency and fixed coupling, relying on a scheduler to avoid crosstalking gates [40]; Google's architectures generally use tunable qubits with either fixed coupler [3] or tunable coupler [2].

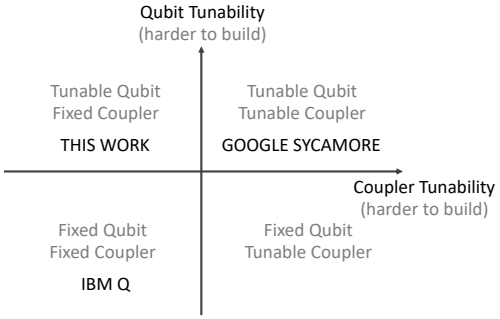

Fig. 1. Technological design choices for mitigating crosstalk. Higher tunability offers better control over the device, but induces higher fabrication overhead and sensitivity to control noise. Our work targets a balanced design, i.e. tunable qubits and fixed coupler, to achieve high program success rate via software optimization of error mitigation.

Crosstalk noise is found to be highly dependent on the interaction strength between the qubits. For instance, Fig. 2 shows the interaction between two connected (directly via a capacitor) frequency-tunable transmon qubits [33]. Unless the two qubit frequencies (ω_A and ω_B) are tuned sufficiently apart, there remains some residual coupling between them, leading to unwanted crosstalk.

When executing a quantum program, qubits are tuned dynamically to their assigned idle and interaction frequencies to perform single-qubit gates and two-qubit gates, respectively. As systems scale up and the frequency range becomes crowded, choosing frequencies for all qubits becomes increasingly challenging, necessitating compiler techniques for tuning frequencies systematically and scheduling instructions intelligently [17].

Fig. 3 is an overview of our approach. This work aims

Corresponding author: yongshan@uchicago.edu

978-1-7281-7383-2/20/$31.00 ©2020 IEEE
DOI 10.1109/MICRO50266.2020.00028

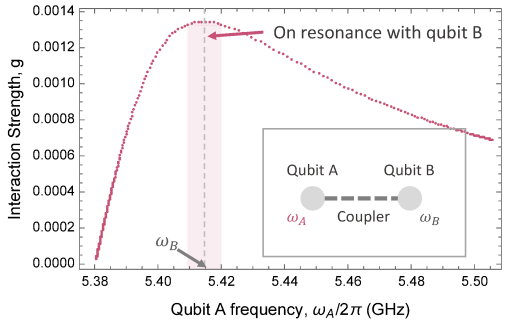

Fig. 2. Interaction strength between two transmon qubits as we tune the frequency ω_A while holding ω_B constant. The strength peaks when two transmons are on resonance ($\omega_A = \omega_B$). Residual coupling remains when ω_A is close to ω_B, and diminishes as ω_A is tuned far away from ω_B. Inset: Schematics of two connected qubits.

to provide means for understanding and mitigating the impact of crosstalk, from a software optimization perspective. Recent work by architects have demonstrated that software optimizations can lead to efficient noise mitigation, effectively providing the equivalent of months of hardware progress. For example, [34], [39], [49] show how to improve qubit utilization, and [21], [48] show how to optimize pulses to speedup gates. We demonstrate that quantum programs can be optimized to reduce the chance of crosstalk and decoherence by scheduling instructions at the right operational frequency and time step, preventing *spectral* and *temporal* collisions, respectively. To do so, we define a type of graph called the *crosstalk graph*; our mitigation technique maps the frequency-aware compilation problem to the coloring of crosstalk graph. Furthermore, the diversity of gate decomposition gives us an extra degree of freedom in scheduling. In sum, our main contributions include:

- An efficient compilation algorithm that mitigates the impact of crosstalk and decoherence via program-specific frequency tuning and instruction scheduling, making tunable-qubit, fixed-coupler systems a competitive, scalable design.
- A systematic analysis of device tunability and sensitivity to provide insights on the advantages and disadvantages of different architectural designs, such as IBM's fixed-frequency systems and Google's tunable-coupler systems.
- Evaluations of our crosstalk mitigation algorithm on a variety of NISQ benchmarks including `BV` [7], `QAOA` [19], `QGAN` [36], `ISING` [6], and `XEB` [2] circuits.

The rest of the paper describes the details of our approach. Section II reviews the superconducting transmon architectures on which this work mainly focuses, and introduces gate operations and crosstalk noises on those architectures. Section III compares the leading superconducting architectures and shows how the frequency-tunable qubit architecture is made competitive by our algorithm. Section IV is our proposed methodology for mitigating the frequency crowding problem, where we define the crosstalk graph and present our frequency tuning

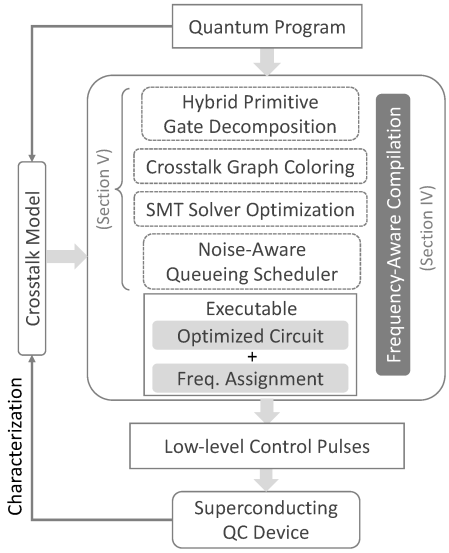

Fig. 3. Flow of our crosstalk mitigation software for tunable superconducting QC systems. We develop a frequency-aware compilation algorithm that systematically reduces crosstalk and decoherence.

algorithm and circuit optimizations. Section V contains the implementation details of the proposed algorithm. Section VI and Section VII evaluates our approach on a suite of NISQ algorithms. Finally, Section VIII discusses the implications and remaining issues.

II. Background

A. Basics of Superconducting Qubits

We start with a brief overview of superconducting qubits and how they are manipulated for computation. Transmon-like variety of superconducting qubits [5], [16], [25], [28] are among the most widely deployed quantum computer architectures [2], [33], [44]. The discussions in this work are centered around techniques for frequency-tunable transmons [4], [5], [6], [28], but some general principles will be applicable to all types of superconducting architectures.

A superconducting transmon *quantum bit* (qubit), as shown in Fig. 4, is by design a multi-level quantum system made out of lithographically printed circuit elements, configured such that they exhibit atom-like energy spectra. The lowest two levels are used as the bit 0 and 1 for computation. The ground energy level represents the state $|0\rangle \equiv [1\ 0]^T$, and the first excited energy level represents the state $|1\rangle \equiv [0\ 1]^T$. Unlike a classical bit, a qubit can be in a linear combination of 0 and 1: $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = [\alpha\ \beta]^T$, where α, β are complex coefficients satisfying $|\alpha|^2 + |\beta|^2 = 1$.

When a transmon gets accidentally excited to the second (or higher) energy level, e.g. $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle + \gamma\,|2\rangle$, for $\gamma \neq 0$, we call this process "leakage". This can happen due to imprecision in quantum control. The energy gap between the ground state $|0\rangle$ and the first excited state $|1\rangle$ is known as the *qubit frequency*, i.e. $\omega_q \equiv \omega_{01} = E_{01}/h$, where h is the

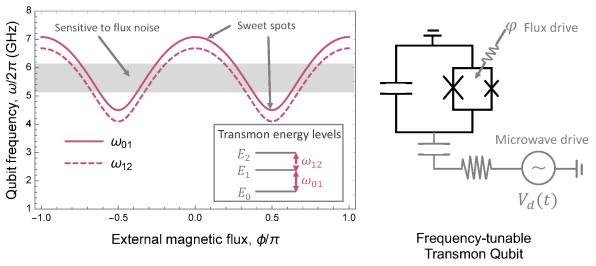

Fig. 4. **Left:** Qubit frequencies as a function of external magnetic flux. The first three levels of the transmon, ω_{01} and ω_{12}, are plotted. Shaded area is where the qubit is sensitive to flux noise. **Right:** Circuit diagram for a frequency-tunable (asymmetric) transmon qubit (highlighted in black), consisting of a capacitor and two asymmetric Josephson junctions. Highlighted in gray are two control lines: the external magnetic flux control φ and microwave voltage drive line $V_d(t)$ for each transmon qubit.

Planck's constant. Hence, we will sometimes use the terms energy and frequency interchangeably. More generally, ω_{01} is referred to as the (first-level) qubit frequency and ω_{02} is the second-level qubit frequency, defined as the gap between the ground state $|0\rangle$ and the second excited state $|2\rangle$. The frequency of a transmon qubit can be changed by applying external magnetic flux through the transmon loop, as shown in Fig. 4. In this case, there are two frequency sweet spots, i.e. frequency values that are relatively stable against flux noise [33]. As such, choosing operating frequencies around the sweet spots is desirable for tunable architectures.

B. Operations and Noises

In QC systems, computation is accomplished by applying a sequence of instructions/operations called *quantum gates*, which take one quantum state to another through unitary transformations, i.e. $|\psi\rangle \rightarrow U|\psi\rangle$, where U is a unitary matrix. These primitive transformations are implemented by driving the qubits via i) microwave voltage signals, and ii) local magnetic flux pulses. The control mechanism for each qubit is illustrated in Fig. 4.

A *quantum compiler* takes a quantum program written in a high-level programming language, performs a series of transformations and optimizations on the intermediate representations (IR) or quantum circuits, and finally outputs low-level control pulses for driving the qubits. At the end, results of the application are obtained by readout operations (called measurements) on the qubits, which collapse each qubit's quantum state to a classical bit $|0\rangle$ or $|1\rangle$.

1) Single-qubit Gates and Decoherence Noise: In superconducting transmon systems, single-qubit gates are implemented by driving the target qubit via: i) a microwave drive line (feeding time-dependent voltage signals $V_d(t)$) through a capacitor connected to the qubit, and ii) a flux drive line (with time-dependent magnetic flux pulses) [33]. For example, `Rx` and `Ry` rotation gates are implemented by sending microwave voltage signals in-phase (I) and out-of-phase (Q) through the drive line, respectively. Other single-qubit gates, such as

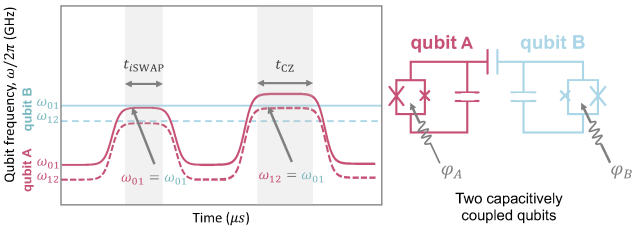

Fig. 5. Two-qubit interactions for two capacitively coupled transmons. **Left:** Two-qubit gates are implemented with resonance of qubit frequencies. Shown here are how qubit frequencies are tuned for i`SWAP` gate and `CZ` gate. **Right:** Circuit diagram of two capacitively coupled transmon qubits.

Hadamard gate (`H`), can be accomplished by a combination of `Rx` and `Ry` gates.

Qubits naturally decay due to perturbations from the environment. Such decay can happen in two ways: i) T_1 *relaxation* (i.e. spontaneous loss of energy causing decay from $|1\rangle$ to $|0\rangle$), and ii) T_2 *dephasing* (i.e. loss of relative quantum phase between $|0\rangle$ and $|1\rangle$). We can model both decays in a combined *decoherence error*: $\epsilon_q(t) = (1 - e^{-t/T_1})(1 - e^{-t/T_2})$, where t is time, and T_1, T_2 are constants characterizing the speed of the decays, for some qubit q.

2) Two-qubit Gates and Crosstalk Noise: Two-qubit gates play important roles in quantum computation, as they implement entangling operations, that is, transformations of one qubit conditioned on the state of the other qubit [10]. Some commonly used two-qubit gates include `CNOT` (controlled-not) gate and `SWAP` gate. Despite their simple forms in the unitary matrix representations, these gates are not typically supported directly in the target architecture. For example, they need to be *decomposed* into primitive gates, such as i`SWAP` gate and `CZ` (controlled-phase) gate, for tunable transmon architectures. The matrix forms for the i`SWAP` gate and the `CZ` gate are:

$$i\texttt{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -i & 0 \\ 0 & -i & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \texttt{CZ} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}.$$

These gates are implemented by tuning the frequencies of the two interacting qubits to some desired operating point, denoted *interaction frequencies*. Then, the qubits are held at that frequency for a duration of time t, depending on the interaction strength g between the two qubits. Fig. 5 depicts this process. Appendix C explains the overhead of dynamically changing qubit frequencies.

In the most general sense, *crosstalk* (i.e. unwanted interaction) happens when two qubits are accidentally tuned on (or close to) resonance. Fig. 2 shows how interaction strength varies with closeness of frequencies, $\delta\omega = |\omega_A - \omega_B|$. Gate time t is shorter when g is higher (i.e. when $\delta\omega$ is small). Two-qubit gate error can therefore be modeled as a function of qubit frequencies and time: $\epsilon_g(\omega, t)$, for any gate g (see Appendix B for details). For example, crosstalk can occur

when a pair of two-qubit gates (on connected qubits simultaneously) happened to use very close interaction frequencies, as highlighted in Fig. 6. Section IV illustrates in details how to understand and mitigate these types of crosstalk error.

III. Related Work

A number of hardware features have been proposed to help mitigate crosstalk: i) connectivity reduction, ii) qubit frequency tuning, and iii) coupler tuning. In addition to these hardware features, some software constraints are usually imposed to effectively reduce crosstalk; for example, certain operations may be prohibited to occur simultaneously.

Connectivity reduction works by building devices with sparse connections between qubits, hence reducing the number of possible crosstalk channels. This greatly increases the circuit mapping and re-mapping overhead for executing a logical circuit, since many `SWAP` gates are needed. Moreover, this model necessitates an intelligent scheduler to serialize operations to avoid crosstalk [40]. This strategy is commonly deployed for fixed-frequency transmon architectures, e.g. from IBM [26]. Because of their non-tunable nature, these architectures have stringent constraints on the initial qubit frequency; a number of optimizers are proposed for this issue [9], [35].

A second class of techniques rely on actively *tuning qubit frequencies* to avoid crosstalk, featured in some prototypes [25] and by Google [3]. Software can decide when to schedule an instruction and which frequency to operate the instruction at. In this class, [50] found a frequency assignment for the surface code circuit; [24] suggests a sudoku-style pattern of frequency assignment for cavity grid.

A third class builds not only frequency-tunable qubits but also *tunable couplers* between qubits, termed "gmon" architectures [11]. Without resorting to permanently reducing device connectivity in hardware, a different subset of connections are activated (via flux drives to the couplers) at different time steps. As such, a schedule for when to activate couplers is needed. After this work is submitted, [31] outlines the frequency optimizer used in [2]. Our results show comparable performance to [31] but with simpler hardware (no tunable couplers). The control parameters used in [31] are hard to predict, but in our evaluation, we include most of the leading noises, e.g., decoherence, sidebands resonance, leakages, flux noises, time overheads of flux tuning, etc.

Most previous studies on quantum program compilation [20], [48] have largely targeted short program execution time (i.e. low circuit depth), and neglected the impact of gate errors such as crosstalk. Optimizations are performed at the gate level, typically involving strategic qubit mapping and instruction scheduling. Recent efforts [35], [40] are among the first to explore architects' role in mitigating crosstalk.

Our work here shows that *frequency-tunable architecture without connectivity reduction and without tunable couplers (but with our software crosstalk mitigation) is competitive against other architectures*. The frequency-tunable but untunable coupler architecture is an optimization sweet spot. On one end of the spectrum, fixed-frequency architectures have a relatively constrained space for software optimization. On the other end of the spectrum, requiring both qubit frequencies and couplers to be tunable introduces higher overhead in fabrication and higher control noises.

IV. Systematic Crosstalk Mitigation

This work aims to demonstrate that *systematic software optimizations* can dramatically mitigate crosstalk, utilizing a variety of microarchitecture tunability features. These features (such as different degree of tunability in qubits themselves and their couplers) allow the hardware to be *dynamically configured* to avoid crosstalk as program executes. We propose frequency-aware software that reduces the chances of both decoherence and crosstalk, via strategic frequency tuning and instruction scheduling.

A. Understanding Crosstalk Constraints

Crosstalk mitigation is one of the major challenges in scaling up superconducting quantum architectures. Each qubit has a frequency ω_q^{01}, as well as its associated higher-level excitation frequency ω_q^{12}, which is slightly smaller than ω_q^{01}. For qubit A and qubit B connected by a capacitor:

(i) when qubits are non-interacting (i.e. during `Identity` or single-qubit gates), their *idle frequencies* should have sufficient separation (e.g. $\omega_A^{01} \neq \omega_B^{01}$, $\omega_A^{01} \neq \omega_B^{12}$, and $\omega_A^{12} \neq \omega_B^{01}$);

(ii) when implementing two-qubit gates, they should be placed on resonance at *interaction frequency* (e.g. $\omega_A^{01} = \omega_B^{01}$ for `iSWAP` gate, and $\omega_A^{01} = \omega_B^{12}$ or $\omega_A^{12} = \omega_B^{01}$ for `CZ` gate).

To avoid crosstalk, every pair of connected qubits must be fabricated or tuned to idle frequencies that satisfy the above constraints. However, each qubit can choose from a limited range [1] of frequency spectrum. Furthermore, every two-qubit gate needs an interaction frequency far enough from those of its neighboring gates. This issue is termed *frequency crowding*, because the frequencies grow increasingly crowded and the above constraints become harder to satisfy, as systems scale up and as programs use more parallelism. It is critical to determine the assignment of frequencies that minimizes unwanted crosstalk.

B. Frequency Tuning and Instruction Scheduling

To remedy this frequency crowding issue, we present a systematic scheme that dynamically tunes the device and schedules instructions according to input programs. Consider the toy program in Fig. 6 as an example – we found that *a general recipe for avoiding crosstalk between two parallel gates is to create sufficient separation: i) either in frequency, ii) or in time.*

In order to understand and mitigate the impact of crosstalk, we begin with two simple observations: i) Every qubit (when not interacting with others) needs to pick a 0-1 excitation frequency sufficiently far apart from the 0-1 or 1-2 excitation

[1]For example, in a typical frequency-tunable transmon architecture, each qubit can be tuned to frequency around 5 GHz to 7 GHz [2].

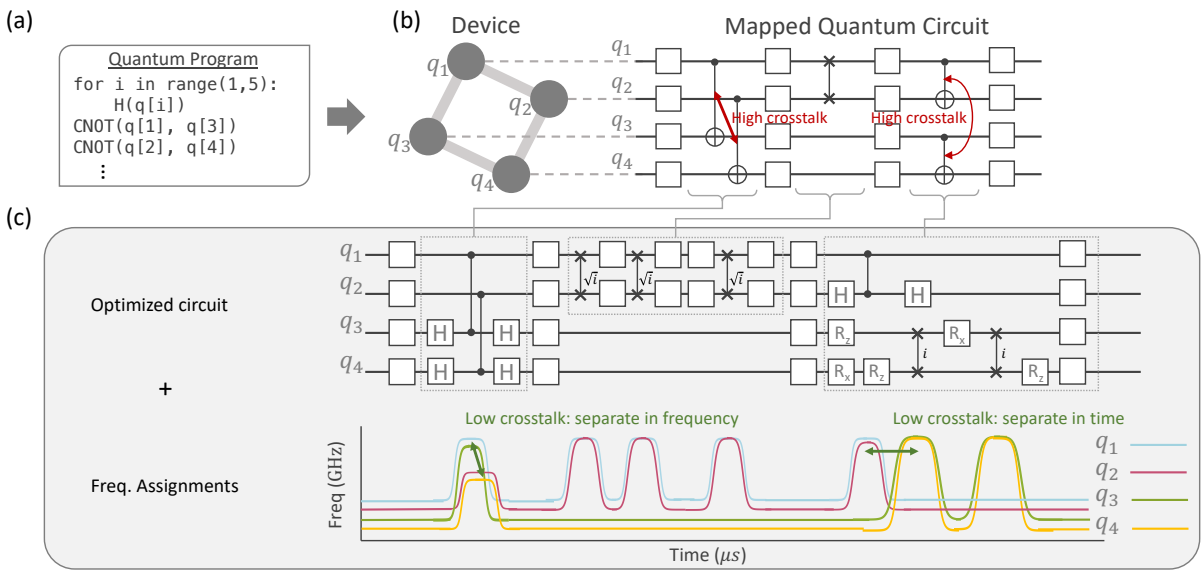

Fig. 6. **(a)** An example quantum program on four qubits. **(b)** The quantum program is mapped to a QC system of 2×2 qubits with nearest-neighbor connectivity. In a quantum circuit, qubits are lines; gates are applied to the qubits from left to right. Highlighted in red are the parallel quantum gates with high likelihood of crosstalk. **(c)** The optimized circuit and frequency assignment resulting from our compilation algorithm. Crosstalk is mitigated by avoiding spectral and temporal collisions in the those gates.

frequencies of its neighbors. ii) The extend of tunability is limited and there are few preferred operational frequencies for each qubit. These two constraints are naturally in tension with each other. The key is to balance the two.

To the best of our knowledge, this work is the first to study strategies for systematically tuning qubit frequencies in a program-aware fashion.

Throughout the remainder of this paper, we explore crosstalk on a flux-tunable transmon architecture with 2-D mesh-like connectivity. Nonetheless, the input to our algorithm can be any arbitrary device topology; hence the crosstalk mitigation techniques we introduce here are applicable to all types of device connectivity, as showed quantitatively in Section VII-F.

C. Resolving Frequency Crowding via Graph Coloring

This section will focus on two types of graphs: i) the device connectivity graph, and ii) the crosstalk graph. For each of these two graphs, we will define formally and illustrate how coloring them can effectively reduce crowding of qubit frequencies.

1) Idle Frequencies and Connectivity Graph: Qubit connectivity is an important characteristic of a quantum device, as it describes the pairs of qubits between which a two-qubit gate can be directly performed. For completeness, we revisit the definition of a connectivity graph: In a connectivity graph G_c, each vertex is a qubit, every edge is a coupling between the two qubits, e.g. a capacitor in the frequency-tunable transmon architecture.

When the qubits are idle (i.e. not interacting with any other qubits), we want to avoid collision of frequencies for every pair of connected qubits. Therefore, we park the qubits at "idle frequencies". To avoid collisions in idle frequencies, it is equivalent to coloring the connectivity graph where no two end-points of an edge share the same color. If a connectivity graph is colorable by c colors, then we need only c frequency values $\{\omega_0, \omega_1, \ldots, \omega_{c-1}\}$ to keep idle qubits from interacting If the separation between the c frequencies are large enough (i.e. any $|\omega_i - \omega_j|$ sufficiently larger than the anharmonicity), then the higher-energy excitation frequencies are also well separated from the other frequencies, reducing interactions through the leakage channel as well. This strategy works well for simple connectivity graphs like the 2-D mesh, because the 2-D mesh is bipartite and thus 2-colorable. We also test the general applicability of our algorithm on different choices of device connectivity.

2) Interaction Frequencies and Crosstalk Graph: Two-qubit gates are implemented by bringing the two qubits on resonance at some "interaction frequency". Any other qubits nearby should be tuned off-resonance from that frequency to avoid unwanted interactions. We define the crosstalk graph to exactly match this constraint. The crosstalk graph G_x of a connectivity graph G_c represent the potential crosstalk that could happen between qubits, which must be addressed by frequency tuning. Here we describe how to construct the crosstalk graph G_x:

(i) Derive the line graph[2] G_L of the connectivity graph G_c.
(ii) Connect two vertices in G_L if the corresponding two

[2]A line graph of a graph G maps each edge in G to a vertex, and two vertices are connected if the two edges in G share a same vertex. [23]

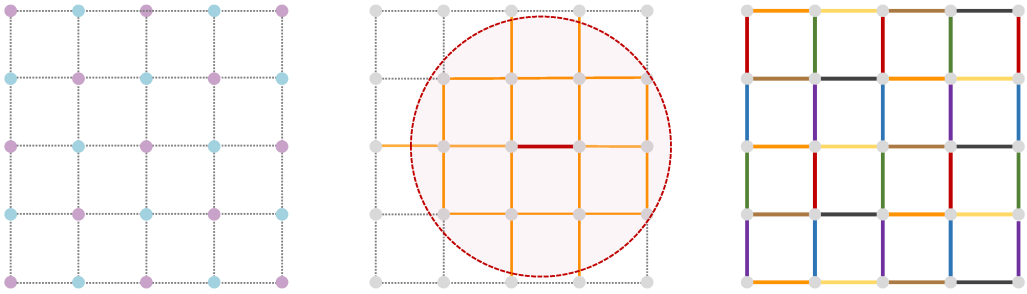

Fig. 7. **Left:** the connectivity graph for a 5×5 mesh of qubits; 2 colors (highlighted in blue and purple) are needed to color the nodes of the graph. The colors map to idle frequencies of the qubits. **Center:** when the two qubits at the center choose an interaction frequency (highlighted in red) all qubits within the crosstalk range must be tuned off resonance from this interaction frequency. **Right:** A non-crosstalking edge coloring of the 2-D mesh, resulting from coloring the crosstalk graph. 8 colors are required to avoid crosstalk among maximum simultaneous operations. Notably, fewer colors will suffice for program-specific compilation that utilizes circuit slicing and subgraph coloring.

edges in G_c is distance[3] one apart.

To elucidate the structures behind the crosstalk graph, we use a 5×5 quantum chip as an example. Consider the middle edge highlighted in red in the center panel of Fig. 7. Every orange edge either shares a common vertex with the red edge or is connected to the red edge by a third edge. Thus in the crosstalk graph, the vertex corresponding to the red edge in G_c is connected with the vertices corresponding to all orange edges. If we tune the qubits on the red edge to an interaction frequency ω_{int}, then during the gate time, none of the orange edges should share that frequency.

Although quite dense (see Fig. 14), the crosstalk graph for a 2-D mesh can be colored by 8 colors as shown at the right of Fig. 7. This coloring is general for any $N \times N$ 2-D mesh, and 8 is the minimum number of colors needed. See Appendix A for an example of idle and interaction frequencies resulting from coloring crosstalk graph.

We report an important observation here: for a device with 2-D mesh connectivity, crosstalk due to frequency crowding is *mostly localized*. In other words, the frequency space does not become more and more crowded as we increase the size of the mesh. To understand how localized is it, we extend our discussion on nearest-neighbor crosstalk to *next*-neighbor crosstalk.

3) Generalization to Higher Distance: So far, we have been discussing crosstalk between directly coupled qubits (i.e. nearest-neighbor crosstalk). One could imagine the residual coupling between a qubit and its next-neighbor could result in crosstalk as well. We introduce a generalization to the crosstalk graph to higher distance d, denoted as $G_x^{(d)}$: The distance-d crosstalk graph $G_x^{(d)}$ of a connectivity graph G_c has a vertex for each edge in G_c, and two vertices are connected if the two edges in G_c share a common vertex or are connected by a path of length d.

[3]Distance between two edges equals the length of the shortest path that connects the two edges.

V. Our Approach

A. Frequency-Aware Compilation: Overview

Now we illustrate the key steps in our crosstalk mitigation algorithm – the inputs to the algorithm include device characteristics (e.g. qubit number, connectivity, transmon tunability), program characteristics (e.g. a scheduled quantum circuit), and optimization level (e.g. crosstalk distance).

Finding optimal (idle and interaction) frequency configurations based on device and program characteristics is a high-dimensional optimization problem; *we break the problem into multiple scalable sub-problems*. As shown in Fig. 3, we begin by constructing a crosstalk graph for the input device. Next, the input program is decomposed into primitive gates and sliced into layers (time steps). Then, we produce *a feasible coloring of an active subgraph* of the crosstalk graph for each layer of the circuit. From the colors, we thereafter map to the idle and interaction frequencies via a Satisfiability Modulo Theory (SMT) solver [8], [15]. Lastly, we produce a feasible schedule of the program (i.e. gate instructions and qubit frequencies for each time step), throttling parallelism if necessary. Algorithm 1 is the main algorithm outlining this process. Specifically, line 10-16 is the queueing schedule in Section V-B6; line 17-19 is the coloring step in Section V-B2; line 20-22 corresponds to the SMT solver optimization in V-B3.

B. Optimization Details

This section is dedicated to explaining the key ingredients of the algorithm in greater detail. Through a series of optimizations, our frequency-aware compilation algorithm drastically reduces the chance of crosstalk and scales favorably with systems sizes, making it a viable long-term solution to frequency tuning for superconducting qubits.

1) Crosstalk Graph Construction: In Section IV-C, we outlined how the crosstalk graph is constructed; the steps are made rigorous in the following Algorithm 2. By abstracting all possible crosstalk channels between pairs of qubits as graph theoretical objects, we are now equipped to quantitatively

Algorithm 1 Frequency-Aware Compilation

$d \leftarrow$ crosstalk distance parameter
$G_c \leftarrow$ connectivity graph of the device D
$G \leftarrow$ gen_crosstalk_graph(D, d)
$C_c \leftarrow$ coloring(G_c)
$\Omega_c \leftarrow$ colors in C_c are mapped to parking frequencies
$P \leftarrow$ decompose input program P into primitive gates
$S \leftarrow$ first layer (time step) of program P
$Q \leftarrow \varnothing$
while S non empty **do**
    $I \leftarrow \varnothing$
    $S \leftarrow$ sort S by criticality
    for gate in S **do**
        if not noise_conflict(gate, I) **then**
            $I \leftarrow I \cup \{gate\}$
        end if
    end for
    $E \leftarrow$ collect relevant two-qubit gates in I
    $H \leftarrow$ subgraph(G, E)
    $C \leftarrow$ coloring(H)
    $\Omega \leftarrow$ smt_find(C)
    $S \leftarrow (S \setminus I) \cup \{$next layer of $P\}$
    $F \leftarrow$ qubit frequencies for this cycle based on Ω_c and Ω
    $Q \leftarrow Q \cup \{(I, F)\}$
end while
return Q

Algorithm 2 gen_crosstalk_graph

$G_c \leftarrow$ connectivity graph of the device D
$G \leftarrow$ networkx.line_graph(G_c)
$S \leftarrow \varnothing$
for pair of nodes (e_1, e_2) in G_ℓ **do**
    $(u_1, v_1) \leftarrow$ pair of qubits for e_1
    $(u_2, v_2) \leftarrow$ pair of qubits for e_2
    cond $\leftarrow dist(u_1, u_2) \leq d$ or $dist(u_1, v_2) \leq d$
    cond $\leftarrow$ cond or $dist(v_1, u_2) \leq d$ or $dist(v_1, v_2) \leq d$
    if cond **then**
        $S \leftarrow S \cup \{(e_1, e_2)\}$
    end if
end for
G.add_edges_from(S)
return G

analyze and systematically mitigate crosstalk errors due to frequency crowding.

2) Circuit Slicing and Subgraph Coloring: One of the major advantages of our approach is in producing a *dynamic* frequency assignment tailored for each input program. This wins over a static (program independent) frequency assignment because frequencies are substantially less crowded when only considering a subset of couplings between qubits that are "active" for a given time step. Here active couplings refers to only those pairs of qubits currently involved in two-qubit gates.

We identify the active subgraph H of the crosstalk graph G, by profiling the two-qubit gates in one time step. The (vertex) coloring of H, denoted as C, is an assignment of labels (called colors) for the vertices of H such that no two adjacent vertices share the same color, while minimizing the number of colors in total. Graph coloring is known to be an NP-complete problem; section VII-C shows how we maintained efficiency. In our optimization, we apply a polynomial-time greedy approximation, the Welsh-Powell algorithm [51], to color the active subgraph.

As a result, a feasible coloring of H yields a set of non-colliding interaction frequencies for the two-qubit gates. Qubits that undergo `Identity` or single-qubit gates are parked at idle frequencies, determined by coloring the device connectivity graph. In the next section, we describe how to map from a coloring to a frequency assignment via a SMT solver.

3) SMT Solver Optimization: The mapping from colors C to frequencies Ω is reduced to a constrained optimization problem. The objective is to assign $|C|$ frequencies within some range $[\omega_{lo}, \omega_{hi}]$, satisfying the crosstalk constraints in Section IV-A. We use a SMT solver to find a feasible solution with the following constraints.

$$\forall c \in C, \omega_{lo} \leq x_c \leq \omega_{hi}, \tag{1}$$

$$\forall x_{c_i}, x_{c_j}, |x_{c_i} - x_{c_j}| \geq \delta, \tag{2}$$

$$|x_{c_i} + \alpha - x_{c_j}| \geq \delta, \tag{3}$$

where α is the anharmonicity, and δ is a threshold. Then, `smt_find` uses a simple binary search to find the maximum threshold δ, for which a feasible solution exists. We ensure the efficiency of the procedure by keeping $|C|$ small.

Once the optimal solution is found, a one-to-one mapping from C to Ω is enforced by a total ordering, motivated by the fact that higher interaction frequency value would yield faster gate time, i.e., $t_{gate} \sim 1/\omega$ [33]. In particular, let us denote $n(c)$ as the number of times c appear in C and $\omega(c)$ as the frequency value to which c maps. We dictate that, for any $c_i, c_j \in C$, if $n(c_i) \geq n(c_j)$ then $\omega(c_i) \geq \omega(c_j)$. The following section details how the frequency ranges are determined.

4) Frequency Partitioning: We partition the range of tunable frequency spectrum into three regions: interaction region, exclusion region, and parking region. Similar partitioning strategies has been studied for surface code error correction circuits [50]. This allows us to decouple the idle frequency assignment from that of the interaction frequency. For a realistic frequency-tunable transmon, the tunable range is typically just a few GHz. So a reasonable design would use a partition with 1 GHz interaction region, 0.5 GHz exclusion region, and 1 GHz parking region. By this design, no frequency is assigned in the exclusion region (which are most sensitive to flux noise), preventing idle qubits from interacting with iswap/cphase qubits.

The interaction frequencies are determined using the coloring C for H. This is a two-step process. First, each coupling in H (that is a pair of qubits performing a two-qubit gate) gets assigned a color $c \in C$ corresponds to an interaction frequency. Second, qubits that appear in its complement $G \setminus H$ remain in their parking frequencies.

5) Hybrid Circuit Decomposition: To implement a two-qubit gate that is not directly supported by the frequency-tunable transmon architecture, we need to decompose it into a series of native gates. Two commonly used two-qubit gates

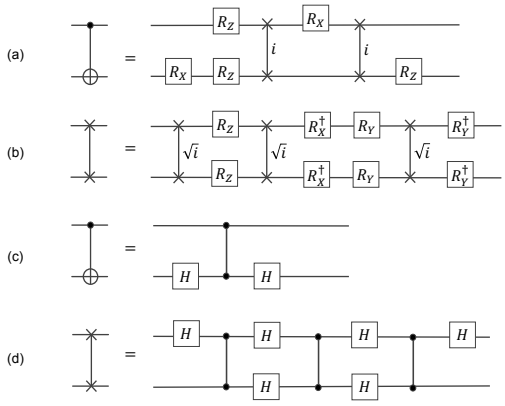

Fig. 8. **(a)**: The CNOT gate, decomposed with iSWAP. **(b)**: The SWAP gate, decomposed with $\sqrt{i\text{SWAP}}$. **(c)**: The CNOT gate, decomposed with CZ. **(d)**: The SWAP gate, decomposed with CZ.

in quantum programs are the CNOT gate and the SWAP gate, because they implement relatively simple Boolean logic. Fig. 8 shows that they can be decomposed into iSWAP (or $\sqrt{i\text{SWAP}}$) and CZ gates.

The strategy for circuit decomposition can affect performance. Compared to decomposing all the two-qubit gates in a circuit with one type of native gates, hybrid strategies can help achieve better fidelity. A simple hybrid strategy is to decompose CNOT gates with CZ, and SWAP gates with $\sqrt{i\text{SWAP}}$. As depicted in Fig. 8, this strategy is advantageous because CNOT (SWAP) is cheaper to implement with CZ ($\sqrt{i\text{SWAP}}$) gates than with $\sqrt{i\text{SWAP}}$ (CZ) gates.

6) Noise-Aware Queueing Scheduler: Finally, parallelism is another crucial concern in our algorithm – on one hand, parallelism helps shorten the circuit execution time, reducing chances of decoherence; on the other hand, it crowds the interaction frequency range, increasing chances of crosstalk. Our noise-aware queueing scheduler finds a sweet spot by strategically serializing gates that are likely to cause crosstalk. In algorithm 1 (line 9-16), gates are delayed based on their criticality and potential noise conflicts. Criticality of a gate is its position along the program critical path, calculated by profiling the input program during circuit slicing on line 7. Function `noise_conflict` predicts potential crosstalk: when scheduling g (e.g. `CNOT(q1,q2)`), if too many of its neighbors in the crosstalk graph are already in I, then their interaction frequencies are likely very close, so we postpone g for the next time step. Serialization is done conservatively while maintaining minimal impact on the critical path length of the program (that is the circuit depth). This *greedy* scheduling approach is shown to be effective in balancing crosstalk and decoherence.

VI. Evaluation

A. Tuning and Scheduling Baselines

We test the performance of our frequency-aware compilation algorithm (i.e. *ColorDynamic*) in comparison to four baselines, *Baseline N* (naive), *Baseline G* (gmon), *Baseline U* (uniform), and *Baseline S* (static), shown in Table I; they represent strategies of frequency tuning and instruction scheduling from leading industry architectures.

TABLE I
List of algorithms used in our evaluation

Algorithms	Microarch. Features
Baseline N	Tunable transmon, fixed coupler, Qiskit [1] scheduler
Baseline G	Tunable transmon, tunable coupler, tiling scheduler
Baseline U	Tunable transmon (with single interaction frequency), fixed coupler, serial scheduler
Baseline S	Tunable transmon, fixed coupler, crosstalk-aware scheduler
ColorDynamic	Tunable transmon, fixed coupler, crosstalk-aware scheduler

Baseline N: Naive Compilation. A conventional crosstalk-unaware compilation algorithm. Qubits are assigned with separated idle and interaction frequencies.

Baseline G: Gmon with Tunable Coupler. This baseline has advanced hardware requirements to activate couplers – the "gmon" architecture, implemented in Google's recent Sycamore quantum architectures [2], takes advantage of both tunable qubit and tunable coupling features to mitigate crosstalk. On the flip side, the flux-tunable coupler would incur fabrication overheads, and introduce extra sensitivity to flux noise. We reconstruct and evaluate a gmon-like architecture where the couplers are activated following the same pattern used for Sycamore, and idle and interaction frequencies match exactly the reported values in [2].

Baseline U: Uniform Frequency with Serialization. This baseline relies on serialization to avoid crosstalk, similar to [26], [40]. All two-qubit gates share one common interaction frequency ω_{int}, demonstrating the impact of serialization.

Baseline S: Static Frequency-Aware Compilation. Baseline S optimizes the idle and interaction frequencies independent of input programs, producing a static set of optimized values. Most crosstalk-aware optimizers perform this type of static optimization [2], [50].

ColorDynamic: Program-specific Frequency-Aware Compilation. This is the pinnacle of our work. Instead of finding a static interaction frequency solution for all programs, ColorDynamic returns optimized frequencies for each time step of a program. It combines all optimizations in Algorithm 1, including circuit slicing, strategical decomposition and serialization, graph coloring, and SMT solvers.

B. Benchmarks

We study the performance of our algorithm through a variety of NISQ benchmarks, shown in Table II. These benchmarks are among the best known applications for near-term quantum machines. We also include circuits for benchmarking simul-

taneous quantum gates to demonstrate the impact of crosstalk on the fidelity of those gates [2].

In our evaluation, we vary number of qubits $n = 4, 9, 16, 25$. These circuits are of most interest, because the range of crosstalk is typically localized, as shown in Fig. 7.

TABLE II
LIST OF BENCHMARKS USED IN OUR EVALUATION

Benchmarks	Descriptions
`BV(n)`	Bernstein-Varzirani (BV) algorithm on n qubits [7]
`QAOA(n)`	Quantum Approximate Optimization Algorithm (QAOA) [19] for MAX-CUT on an Erdos-Renyi random graph with n vertices
`ISING(n)`	Linear Ising model simulation of spin chain of length n [6]
`QGAN(n)`	Quantum Generative Adversarial Network (QGAN) with training data of dimentsion 2^n [36]
`XEB(n, p)`	Cross entropy benchmarking circuit for calibrating two-qubit gates on n qubits with p cycles [2]

C. Experimental Setup

Software implementation: Our compilation algorithms are implemented in `Python` 3.7, interfacing the IBM `Qiskit` software library [1]. The graph coloring optimization uses `greedy_coloring` in `NetworkX` library [22], and the SMT optimization uses `Z3 solver` [15] through the `Z3py` APIs. All compilation experiments use Intel E5-2680v4 (2.4GHz, 64GB RAM).

Architectural features: We consider a 2D grid of $N \times N$ asymmetric frequency-tunable transmons, each having maximum frequencies ω_q (in GHz) sampled from Gaussian distribution: $\Omega \sim \mathcal{N}(\omega, 0.1)$, with nearly constant aharmonicity $\alpha/2\pi = (\omega_{12} - \omega_{01})/2\pi \approx 200$ MHz, to account for realistic variation in fabrication and initial detuning. Any pair of nearest-neighbor qubits are directly connected with a capacitor; the coupling strength g depends on the frequencies of the qubits, which is typically around $g/2\pi \approx 30$ MHz. For gmon-like experiments, qubits are connected by flux-tunable couplers, each with its own independent external magnetic flux control. These parameters are set to realistic values in line with experimental data from the literature [29].

Metrics: For our compilation experiments, we need to efficiently compute the program success rate – we define a heuristic for efficiently estimating the *worst case* success rate of a program under crosstalk and decoherence noises.

$$P_{success} = \Pi_{g \in G}(1 - \epsilon_g) \cdot \Pi_{q \in Q}(1 - \epsilon_q) \quad (4)$$

where ϵ_g is the crosstalk gate error, and ϵ_q is qubit decoherence error. Details on ϵ_g can be found in Appendix B, equation 6; ϵ_q is captured by modeling T_1 and T_2 during idle or gate time, as studied in [29]. A similar metric to $P_{success}$ is used in [2], [53].

Besides being efficiently computable, this heuristic has useful operational significance – we can understand and mitigate the worst-case impact of crosstalk and decoherence on the systems during compile-time or run-time of quantum programs. Of course, to gain full knowledge of the crosstalk and decoherence errors, we need full noisy circuit simulation, which quickly becomes intractable as circuit size grows beyond tens of qubits. Hence, we validate the heuristic estimator on small-scale circuits, for which noisy circuit simulation is possible.

VII. RESULTS

A. Program Success Rate

Fig. 9 shows worst-case overall success rate, estimated using our heuristic equation 4. Note that statistics, such as those from `qaoa(16)` and `ising(16)` circuits, are excluded from the analysis due to their estimated success rates being lower than 10^{-4}. Baseline N is crosstalk-unaware; as a result, crosstalk has detrimental impact on program success rates for any circuit with parallel two-qubit gates on adjacent qubits, as shown in Fig. 9. ColorDynamic achieves *comparable performance to Baseline G but with simpler hardware (no tunable couplers)*. Results for Baseline G in Fig. 9 is a conservative estimate, assuming couplers can be deactivated *perfectly*. We study the effect of residual coupling in Fig. 12. Compared to Baseline U (with serialization), ColorDynamic consistently outperforms, *achieving 13.3x better success rate on average*. Compared to Baseline S, across all benchmarks, ColorDynamic outperforms static strategies because it is able to exploit program structures and assign frequencies tailored for every layer of the program.

B. Impact on Serialization

Fig. 10 compares the resulting program depth and decoherence error across algorithms. Although serialization can effectively prevent gates from crosstalk (commonly adopted such as for IBM's fixed-frequency qubits), it results in deeper circuits (i.e. longer execution time), which consequently implies higher qubit decoherence. Overall, baseline U requires the most amount of serialization. ColorDynamic produces *1.02x average decoherence error*, compared to baseline G, and *0.90x average decoherence error*, compared to baseline U. Lower decoherence error is desirable when executing on NISQ hardware.

C. Scalability and Complexity

Globally optimizing for the best frequency configuration based on device and program characteristics is challenging; our approach breaks the optimization problem into multiple scalable sub-problems. ColorDynamic keeps the complexity of each sub-problem small, trading off program parallelism for optimization complexity when necessary. In particular, the leading costs stem from coloring of crosstalk graphs and application of SMT solvers.

The greedy coloring algorithm takes time polynomially in the graph size, which is kept small thanks to circuit slicing and strategic serialization. The number of variables/constraints in the SMT solver is proportional to the number of colors obtained from coloring; in the next section, we demonstrate that the number of colors remains small. Empirically, we report

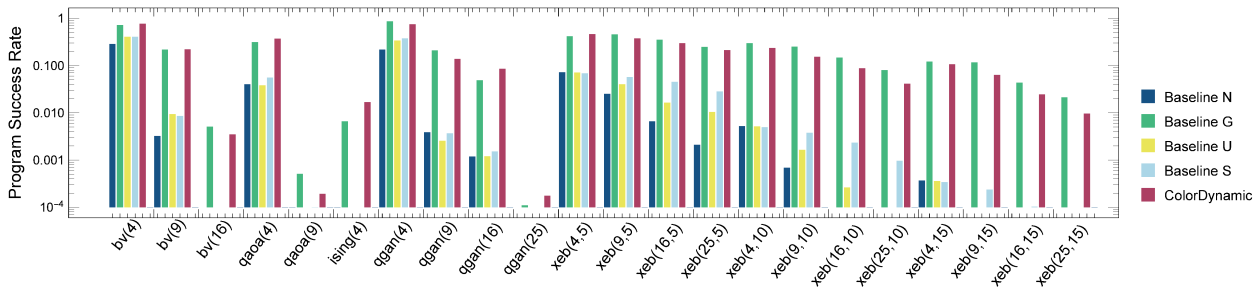

Fig. 9. Log-scale worst-case program success rates using crosstalk-mitigation algorithms, estimated by heuristics. Higher success rate is better. Across the benchmarks, ColorDynamic performs consistently well compared to other algorithms. In particular, it matches the crosstalk resilience of baseline G (with tunable-qubit, tunable coupler), but on fixed-coupler hardware which is more robust to external noise. Results for qaoa(16) and ising(16) are omitted due to high circuit depth and qubit decoherence.

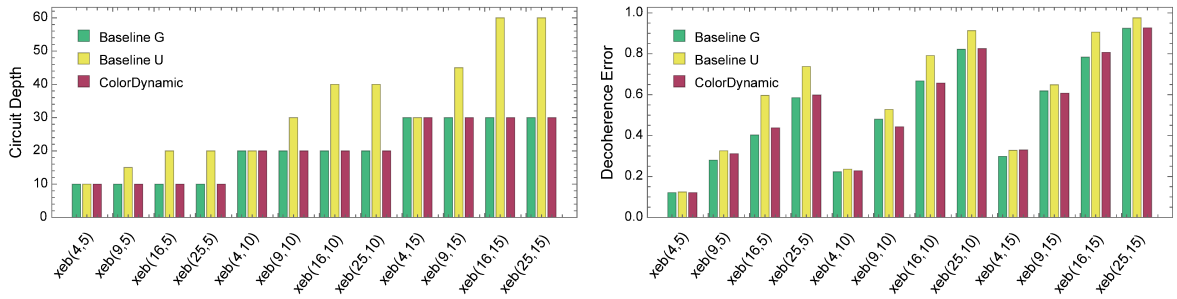

Fig. 10. **Left:** Circuit depth resulting from crosstalk-mitigation algorithms. Across the benchmarks, ColorDynamic avoids crosstalk without incurring significant serialization. **Right:** Decoherence errors resulting from crosstalk-mitigation algorithms. Lower is better.

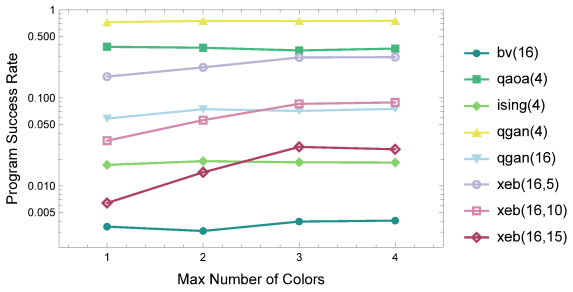

Fig. 11. Finding sweet spot of tunability. More than three colors (i.e. frequencies) are typically *unnecessary* for NISQ benchmarks.

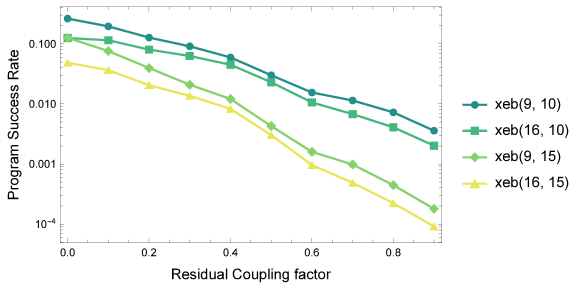

Fig. 12. Log-scale success rate by strength of residual coupling. Baseline G success rate decays exponentially as residual coupling increases.

the *number of colors and compilation time* of ColorDynamic across benchmarks in Fig. 13. Compilation time *remains less than* 30 *seconds on systems up to 81 qubits* for a highly parallel benchmark such as XEB.

D. Sensitivity on Tunability

In ColorDynamic, we can limit the maximum number of colors used for assigning qubit frequencies. To guarantee low crosstalk, fewer colors implies more serialization. In Fig. 11, we examine the balance between spectral and temporal optimizations, and find the best tunability for each benchmark. In general, we observe optimal operating point at 1 or 2 colors, depending on the initial parallelism of the benchmark. This result has significant hardware implications – such program-specific optimization shows that frequency-tunable qubits with 2 frequency sweet spots are good candidates for near-term algorithms, hence building qubits with more sweet spots will only give diminishing returns.

E. Gmon's Sensitivity to Residual Coupling

In our evaluation, Baseline G conservatively assumes that coupling can be (de)activated *perfectly*. In practice, tuning couplers increases sensitivity to control noises. In Fig. 12, we demonstrate how the performance degrade exponentially as residual coupling increases. Such exponential decay in performance motivates the necessity of strategic frequency tuning for tunable qubit and coupler architectures.

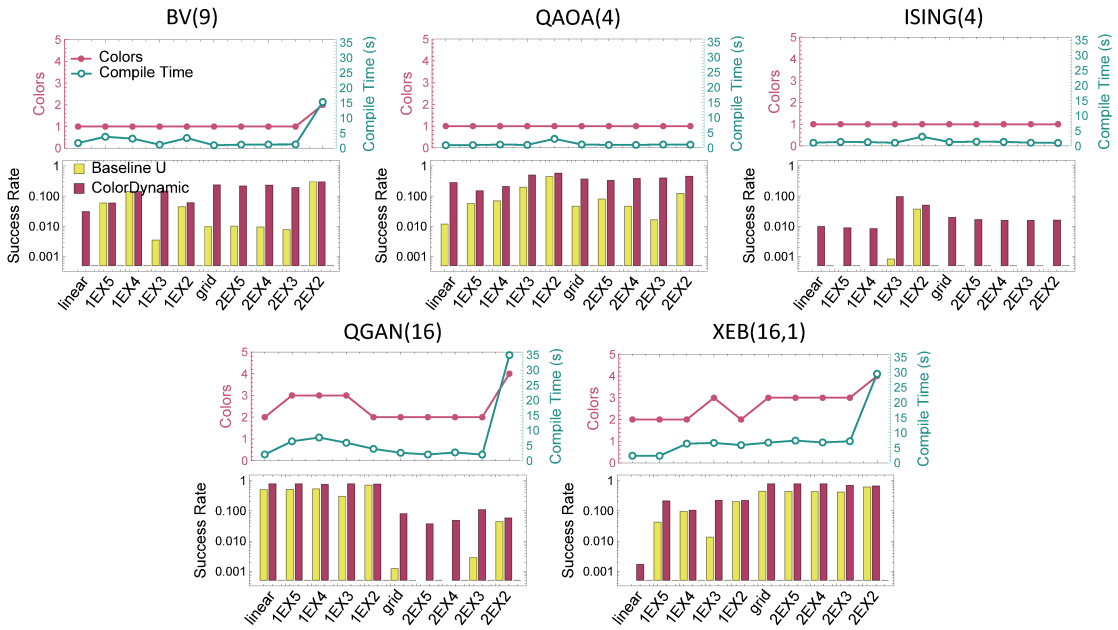

Fig. 13. Results on general device connectivity across benchmarks. **Top:** Number of colors (for interaction frequency) and compilation time of ColorDynamic. **Bottom:** Log-scale program success rate for Baseline U and ColorDynamic. Denser connectivity from left to right along x-axis. n-EX-k is an n-ary express cube [14] with inserted connections every k nodes.

F. General Device Connectivity

To demonstrate the general applicability of our algorithm with respect to device connectivity, we perform a systematic study shown in Fig. 13. Denser connectivity for superconducting device is challenging [30], due to limitations such as coupling and addressing qubits. As such, we target a class of connectivity graphs with increasing density while incurring minimal wiring overhead, namely the "express cubes" [14] designed for interconnection networks. In particular, we augment an increasing number of connections to a 1-D linear path and a 2-D grid, denoted as 1EX-k and 2EX-k graphs respectively, where k stands for inserting a connection every k nodes [14].

ColorDynamic consistently *improves program success rate by 3.97x in geometric mean* across all benchmarks, compared to baseline U. Depending on applications, best performance is usually found on connectivity not too sparse or denser than grid. Compilation time of ColorDynamic is kept low ($\sim$ 10 seconds) in practice, because the number of colors remains small, as argued in Section VII-C and Fig. 11. Empirically, we see some increase in the extreme cases with unrealistically dense connectivity, but still within a desirable range.

VIII. Conclusion

In this work, we introduce a systematic approach to software mitigation of crosstalk due to frequency crowding. Our approach allows fixed coupler architectures to compete with tunable coupler architectures in reliability, potentially simplifying the fabrication of quantum machines. The general applicability of our algorithm with respect to device connectivity also motivates potential paths forward in terms of hardware connectivity design. One extension to our work is to apply the methodology of ColorDynamic to guide both qubit tuning and coupler tuning. In fact, the methodology is extensible to any quantum architectures with tunable qubits; it solves a generic calibration problem for isolating or interacting qubits. Finally, complementing Gmon architecture with ColorDynamic optimization would also be a natural extension.

The compilation and simulation software used in this paper is open-sourced and available on GitHub [18].

Acknowledgments

This work is funded in part by EPiQC, an NSF Expedition in Computing, under grants CCF-1730449; in part by STAQ, under grant NSF Phy-1818914; and in part by DOE grants DE-SC0020289 and DE-SC0020331. This work was completed in part with resources provided by the University of Chicago Research Computing Center. PG is supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program. We thank Kenneth Brown, Ike Chuang, Morten Kjaergaard, Nelson Leung, Prakash Murali, David Schuster, and Christopher Wang for fruitful discussions. We also thank the anonymous reviewers for their valuable comments and suggestions.

Appendix A
Example Idle and Interaction Frequencies by ColorDynamic

This section provides a concrete example of the idle and interaction frequencies for a 4×4 qubit systems, resulting from the proposed ColorDynamic algorithm, as shown in

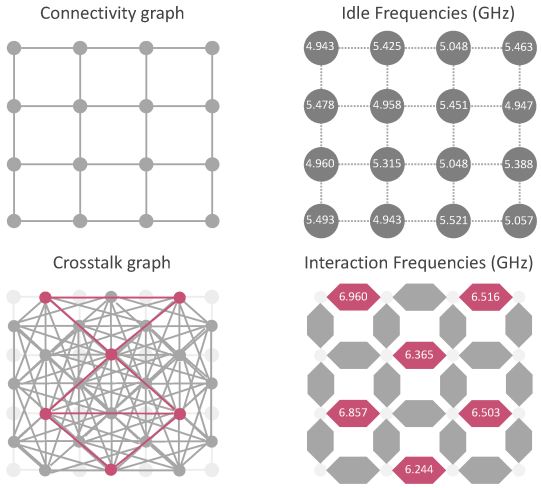

Fig. 14. Example qubit frequencies, ω_{01}. **Top:** the connectivity graph G_c of a 4×4 qubit mesh on the left, and the resulting idle frequencies by coloring G_c on the right. **Bottom:** the crosstalk graph G_x on the left, and the resulting interaction frequencies for the subgraph of G_x highlighted in red on the right.

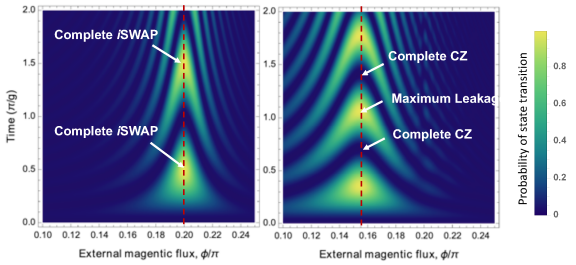

Fig. 15. **Left:** Probability of state transition between $|01\rangle$ and $|10\rangle$, as a function of external magnetic flux and time. **Right:** Probability of state transition between $|11\rangle$ and $|20\rangle$, as a function of external magnetic flux and time.

Fig. 14. Notably, the idle frequencies are assigned in a checker board pattern of high and low values to avoid crosstalk with nearest neighbors. The interaction frequencies are assigned to the qubits performing simultaneous two-qubit gates in one of the time-steps of the `xeb(16, p)` circuit [2]. Frequencies are optimized by subgraph coloring and SMT solvers. Each asymmetric transmon qubit has two sweet spots, as shown in Fig. 4. As such, we keep the idle frequencies close to the low sweet spot near 5 GHz, and the interaction frequencies close to the high sweet spot near 7 GHz.

Appendix B
Gate Errors due to Crosstalk

We continue to elaborate on our heuristic noise model for estimating gate error $\epsilon_g(\omega, t)$, following Section II and VI. For frequency-tunable transmon qubits, two-qubit gates are accomplished via resonance. Depending on the energy levels that the resonance occurs, we can implement i`SWAP` and `CZ` gates. In Fig. 15, we plot the probability of state transitions as we tune the local magnetic flux of one of the qubit (along x-axis) and as the time spent on that operating point is increased (along y-axis). Let $\delta\omega = |\omega_A - \omega_B|$ be the frequency difference of two adjacent qubits, with residual coupling strength [33]:

$$g'(\delta\omega) = \frac{g_0^2}{\hbar^2 \delta\omega}, \tag{5}$$

as shown in Fig. 2, where g_0 depends on the effective coupling capacitance C_{qq}. The coupling strength determines how fast and strong the state transitions undergo. When brought on resonance, the two states $|01\rangle$ and $|10\rangle$ will undergo Rabi oscillation, giving rise to a periodic exchange of energy population. The transition probability is $\Pr[t] = \sin(gt)^2$, where g is the coupling strength. Following [3], the crosstalk error (for idle qubits) is

$$\epsilon_g(\delta\omega, t) = 1 - \sin\left(g'(\delta\omega)t\right)^2. \tag{6}$$

For i`SWAP` gate operations, we want a complete exchange of population, predicted at $t = \frac{\pi}{2g}$. We note that for $t = \frac{\pi}{4g}$, it results in another important operation relevant to this work, the $\sqrt{i\texttt{SWAP}}$ gate. The `CZ` operation is implemented by resonance of $|11\rangle$ and $|20\rangle$. Due to the higher photon number, the coupling strength is scaled by a constant factor, $\sqrt{2}g$. A complete `CZ` happens when exchanged from $|11\rangle$ to $|20\rangle$, and back to $|11\rangle$, in other words, `CZ` gate time is $t = \frac{\pi}{\sqrt{2}g}$.

Appendix C
Overhead of Dynamic Tuning

Our algorithm relies on dynamically changing qubit frequencies via an external magnetic flux (Fig. 4). Our simulation analysis has taken both the *time and error overheads* into account, including flux control noise. Flux tuning has been experimentally demonstrated in fast gate implementations and system calibrations [30]. The time overhead of flux tuning is only a fraction of quantum gate. How fast the pulses are changed is parametrically controlled; state-of-the-art controls show accurate, fast tuning (within 2 ns) [46], giving rise to fast single-qubit flux (`Rz`) gate and two-qubit i`SWAP` and `CZ` gates (around 50 ns) with $> 99.5\%$ fidelity [29], compared to a two-qubit `CR` gate (around 160 ns [47]) on fixed-frequency qubit architectures. Control noises and pulse distortions are indeed present in current tunable systems [2]; techniques such as frequency sweet-spots (Fig. 4) and limiting number of colors (Fig. 11) are designed to mitigate them.

References

[1] H. Abraham, I. Y. Akhalwaya, G. Aleksandrowicz, T. Alexander, G. Alexandrowics, E. Arbel, A. Asfaw, C. Azaustre, AzizNgoueya, P. Barkoutsos, G. Barron, L. Bello, Y. Ben-Haim, D. Bevenius, L. S. Bishop, S. Bosch, S. Bravyi, D. Bucher, F. Cabrera, P. Calpin, L. Capelluto, J. Carballo, G. Carrascal, A. Chen, C.-F. Chen, R. Chen, J. M. Chow, C. Claus, C. Clauss, A. J. Cross, A. W. Cross, S. Cross, J. Cruz-Benito, C. Culver, A. D. Córcoles-Gonzales, S. Dague, T. E. Dandachi, M. Dartiailh, DavideFrr, A. R. Davila, D. Ding, J. Doi, E. Drechsler, Drew, E. Dumitrescu, K. Dumon, I. Duran, K. EL-Safty, E. Eastman, P. Eendebak, D. Egger, M. Everitt, P. M. Fernández, A. H. Ferrera, A. Frisch, A. Fuhrer, M. GEORGE, J. Gacon, Gadi, B. G. Gago, J. M. Gambetta, A. Gammanpila, L. Garcia, S. Garion, J. Gomez-Mosquera, S. de la Puente González, I. Gould, D. Greenberg, D. Grinko, W. Guan, J. A. Gunnels, I. Haide, I. Hamamura, V. Havlicek, J. Hellmers, Ł. Herok, S. Hillmich, H. Horii, C. Howington,

S. Hu, W. Hu, H. Imai, T. Imamichi, K. Ishizaki, R. Iten, T. Itoko, A. Javadi-Abhari, Jessica, K. Johns, T. Kachmann, N. Kanazawa, Kang-Bae, A. Karazeev, P. Kassebaum, S. King, Knabberjoe, A. Kovyrshin, V. Krishnan, K. Krsulich, G. Kus, R. LaRose, R. Lambert, J. Latone, S. Lawrence, D. Liu, P. Liu, Y. Maeng, A. Malyshev, J. Marecek, M. Marques, D. Mathews, A. Matsuo, D. T. McClure, C. McGarry, D. McKay, D. McPherson, S. Meesala, M. Mevissen, A. Mezzacapo, R. Midha, Z. Minev, A. Mitchell, N. Moll, M. D. Mooring, R. Morales, N. Moran, P. Murali, J. Müggenburg, D. Nadlinger, G. Nannicini, P. Nation, Y. Naveh, P. Neuweiler, P. Niroula, H. Norlen, L. J. O'Riordan, O. Ogunbayo, P. Ollitrault, S. Oud, D. Padilha, H. Paik, S. Perriello, A. Phan, M. Pistoia, A. Pozas-iKerstjens, V. Prutyanov, D. Puzzuoli, J. Pérez, Quintiii, R. Raymond, R. M.-C. Redondo, M. Reuter, J. Rice, D. M. Rodríguez, M. Rossmannek, M. Ryu, T. SAPV, SamFerracin, M. Sandberg, N. Sathaye, B. Schmitt, C. Schnabel, Z. Schoenfeld, T. L. Scholten, E. Schoute, J. Schwarm, I. F. Sertage, K. Setia, N. Shammah, Y. Shi, A. Silva, A. Simonetto, N. Singstock, Y. Siraichi, I. Sitdikov, S. Sivarajah, M. B. Sletfjerding, J. A. Smolin, M. Soeken, I. O. Sokolov, SooluThomas, D. Steenken, M. Stypulkoski, J. Suen, H. Takahashi, I. Tavernelli, C. Taylor, P. Taylour, S. Thomas, M. Tillet, M. Tod, E. de la Torre, K. Trabing, M. Treinish, TrishaPe, W. Turner, Y. Vaknin, C. R. Valcarce, F. Varchon, A. C. Vazquez, D. Vogt-Lee, C. Vuillot, J. Weaver, R. Wieczorek, J. A. Wildstrom, R. Wille, E. Winston, J. J. Woehr, S. Woerner, R. Woo, C. J. Wood, R. Wood, S. Wood, J. Wootton, D. Yeralin, R. Young, J. Yu, C. Zachow, L. Zdanski, C. Zoufal, Zoufalc, azulehner, bcamorrison, brandhsn, chlorophyll zz, dan1pal, dime10, drholmie, elfrocampeador, faisaldebouni, fanizzamarco, gruu, kanejess, klinvill, kurarrr, lerongil, ma5x, merav aharoni, ordmoj, sethmerkel, strickroman, sumitpuri, tigerjack, toural, vvilpas, welien, willhbang, yang.luh, yelojakit, and yotamvaknin ibm, "Qiskit: An open-source framework for quantum computing," 2019.

[2] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[3] R. Barends, C. Quintana, A. Petukhov, Y. Chen, D. Kafri, K. Kechedzhi, R. Collins, O. Naaman, S. Boixo, F. Arute *et al.*, "Diabatic gates for frequency-tunable superconducting qubits," *Physical Review Letters*, vol. 123, no. 21, p. 210501, 2019.

[4] R. Barends, J. Kelly, A. Megrant, D. Sank, E. Jeffrey, Y. Chen, Y. Yin, B. Chiaro, J. Mutus, C. Neill *et al.*, "Coherent josephson qubit suitable for scalable quantum integrated circuits," *Physical review letters*, vol. 111, no. 8, p. 080502, 2013.

[5] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell *et al.*, "Superconducting quantum circuits at the surface code threshold for fault tolerance," *Nature*, vol. 508, no. 7497, pp. 500–503, 2014.

[6] R. Barends, A. Shabani, L. Lamata, J. Kelly, A. Mezzacapo, U. Las Heras, R. Babbush, A. G. Fowler, B. Campbell, Y. Chen *et al.*, "Digitized adiabatic quantum computing with a superconducting circuit," *Nature*, vol. 534, no. 7606, pp. 222–226, 2016.

[7] E. Bernstein and U. Vazirani, "Quantum complexity theory," *SIAM Journal on computing*, vol. 26, no. 5, pp. 1411–1473, 1997.

[8] N. Bjørner, A.-D. Phan, and L. Fleckenstein, "νz-an optimizing smt solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2015, pp. 194–199.

[9] M. Brink, J. M. Chow, J. Hertzberg, E. Magesan, and S. Rosenblatt, "Device challenges for near term superconducting quantum processors: frequency collisions," in *2018 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2018, pp. 6–1.

[10] S. Caldwell, N. Didier, C. Ryan, E. Sete, A. Hudson, P. Karalekas, R. Manenti, M. da Silva, R. Sinclair, E. Acala *et al.*, "Parametrically activated entangling gates using transmon qubits," *Physical Review Applied*, vol. 10, no. 3, p. 034050, 2018.

[11] Y. Chen, C. Neill, P. Roushan, N. Leung, M. Fang, R. Barends, J. Kelly, B. Campbell, Z. Chen, B. Chiaro *et al.*, "Qubit architecture with high coherence and fast tunable coupling," *Physical review letters*, vol. 113, no. 22, p. 220502, 2014.

[12] A. D. Córcoles, E. Magesan, S. J. Srinivasan, A. W. Cross, M. Steffen, J. M. Gambetta, and J. M. Chow, "Demonstration of a quantum error detection code using a square lattice of four superconducting qubits," *Nature communications*, vol. 6, no. 1, pp. 1–10, 2015.

[13] D. A. Craik, N. Linke, M. Sepiol, T. Harty, J. Goodwin, C. Ballance, D. Stacey, A. Steane, D. Lucas, and D. Allcock, "High-fidelity spatial and polarization addressing of ca+ 43 qubits using near-field microwave control," *Physical Review A*, vol. 95, no. 2, p. 022337, 2017.

[14] W. J. Dally, "Express cubes: Improving the performance ofk-ary n-cube interconnection networks," *IEEE Transactions on Computers*, vol. 40, no. 9, pp. 1016–1023, 1991.

[15] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[16] L. DiCarlo, J. M. Chow, J. M. Gambetta, L. S. Bishop, B. R. Johnson, D. Schuster, J. Majer, A. Blais, L. Frunzio, S. Girvin *et al.*, "Demonstration of two-qubit algorithms with a superconducting quantum processor," *Nature*, vol. 460, no. 7252, pp. 240–244, 2009.

[17] Y. Ding and F. T. Chong, "Quantum computer systems: Research for noisy intermediate-scale quantum computers," *Synthesis Lectures on Computer Architecture*, vol. 15, no. 2, pp. 1–227, 2020.

[18] Y. Ding, P. Gokhale, T. Propson, C. Winkler, and S. F. Lin, "FastSC: Frequency-Aware Synthesis Toolbox for Superconducting Quantum Computers," https://github.com/yongshanding/FastSC, EPiQC, Aug 2020.

[19] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.

[20] P. Gokhale, Y. Ding, T. Propson, C. Winkler, N. Leung, Y. Shi, D. I. Schuster, H. Hoffmann, and F. T. Chong, "Partial compilation of variational algorithms for noisy intermediate-scale quantum machines," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 266–278.

[21] P. Gokhale, A. Javadi-Abhari, N. Earnest, Y. Shi, and F. T. Chong, "Optimized quantum compilation for near-term algorithms with openpulse," *arXiv preprint arXiv:2004.11205*, 2020.

[22] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[23] F. Harary and R. Z. Norman, "Some properties of line digraphs," *Rendiconti del Circolo Matematico di Palermo*, vol. 9, no. 2, pp. 161–168, 1960.

[24] F. Helmer, M. Mariantoni, A. G. Fowler, J. von Delft, E. Solano, and F. Marquardt, "Cavity grid for scalable quantum computation with superconducting circuits," *EPL (Europhysics Letters)*, vol. 85, no. 5, p. 50007, 2009.

[25] M. Hutchings, J. B. Hertzberg, Y. Liu, N. T. Bronn, G. A. Keefe, M. Brink, J. M. Chow, and B. Plourde, "Tunable superconducting qubits with flux-independent coherence," *Physical Review Applied*, vol. 8, no. 4, p. 044003, 2017.

[26] "Quantum Takes Flight: Moving from Laboratory Demonstrations to Building Systems," https://www.ibm.com/blogs/research/2020/01/quantum-volume-32/, accessed: 2020-04-05.

[27] "Intel Introduces 'Horse Ridge' to Enable Commercially Viable Quantum Computers," https://newsroom.intel.com/news/intel-introduces-horse-ridge-enable-commercially-viable-quantum-computers/#gs.2es8bu, accessed: 2020-04-05.

[28] J. Kelly, R. Barends, A. G. Fowler, A. Megrant, E. Jeffrey, T. C. White, D. Sank, J. Y. Mutus, B. Campbell, Y. Chen *et al.*, "State preservation by repetitive error detection in a superconducting quantum circuit," *Nature*, vol. 519, no. 7541, pp. 66–69, 2015.

[29] M. Kjaergaard, M. Schwartz, A. Greene, G. Samach, A. Bengtsson, M. O'Keeffe, C. McNally, J. Braumüller, D. Kim, P. Krantz *et al.*, "A quantum instruction set implemented on a superconducting quantum processor," *arXiv preprint arXiv:2001.08838*, 2020.

[30] M. Kjaergaard, M. E. Schwartz, J. Braumüller, P. Krantz, J. I.-J. Wang, S. Gustavsson, and W. D. Oliver, "Superconducting qubits: Current state of play," *Annual Review of Condensed Matter Physics*, vol. 11, pp. 369–395, 2020.

[31] P. V. Klimov, J. Kelly, J. M. Martinis, and H. Neven, "The snake optimizer for learning quantum processor control parameters," *arXiv preprint arXiv:2006.04594*, 2020.

[32] J. Koch, M. Y. Terri, J. Gambetta, A. A. Houck, D. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf, "Charge-insensitive qubit design derived from the cooper pair box," *Physical Review A*, vol. 76, no. 4, p. 042319, 2007.

[33] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver, "A quantum engineer's guide to superconducting qubits," *Applied Physics Reviews*, vol. 6, no. 2, p. 021318, 2019.

[34] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the Twenty-Fourth*

International Conference on Architectural Support for Programming Languages and Operating Systems, 2019, pp. 1001–1014.
[35] ——, "Towards efficient superconducting quantum processor architecture design," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 1031–1045.
[36] S. Lloyd and C. Weedbrook, "Quantum generative adversarial learning," *Physical review letters*, vol. 121, no. 4, p. 040502, 2018.
[37] D. C. McKay, S. Sheldon, J. A. Smolin, J. M. Chow, and J. M. Gambetta, "Three-qubit randomized benchmarking," *Physical review letters*, vol. 122, no. 20, p. 200502, 2019.
[38] P. Mundada, G. Zhang, T. Hazard, and A. Houck, "Suppression of qubit crosstalk in a tunable coupling superconducting circuit," *Physical Review Applied*, vol. 12, no. 5, p. 054023, 2019.
[39] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 1015–1029.
[40] P. Murali, D. C. McKay, M. Martonosi, and A. Javadi-Abhari, "Software mitigation of crosstalk on noisy intermediate-scale quantum computers," *arXiv preprint arXiv:2001.02826*, 2020.
[41] C. Neill, P. Roushan, K. Kechedzhi, S. Boixo, S. V. Isakov, V. Smelyanskiy, A. Megrant, B. Chiaro, A. Dunsworth, K. Arya *et al.*, "A blueprint for demonstrating quantum supremacy with superconducting qubits," *Science*, vol. 360, no. 6385, pp. 195–199, 2018.
[42] C. Ospelkaus, C. E. Langer, J. M. Amini, K. R. Brown, D. Leibfried, and D. J. Wineland, "Trapped-ion quantum logic gates based on oscillating magnetic fields," *Physical review letters*, vol. 101, no. 9, p. 090502, 2008.
[43] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.
[44] M. Reagor, C. B. Osborn, N. Tezak, A. Staley, G. Prawiroatmodjo, M. Scheer, N. Alidoust, E. A. Sete, N. Didier, M. P. da Silva *et al.*, "Demonstration of universal parametric entangling gates on a multi-qubit lattice," *Science advances*, vol. 4, no. 2, p. eaao3603, 2018.
[45] M. D. Reed, L. DiCarlo, S. E. Nigg, L. Sun, L. Frunzio, S. M. Girvin, and R. J. Schoelkopf, "Realization of three-qubit quantum error correction with superconducting circuits," *Nature*, vol. 482, no. 7385, pp. 382–385, 2012.
[46] M. Rol, F. Battistel, F. Malinowski, C. Bultink, B. Tarasinski, R. Vollmer, N. Haider, N. Muthusubramanian, A. Bruno, B. Terhal *et al.*, "Fast, high-fidelity conditional-phase gate exploiting leakage interference in weakly anharmonic superconducting qubits," *Physical review letters*, vol. 123, no. 12, p. 120502, 2019.
[47] S. Sheldon, E. Magesan, J. M. Chow, and J. M. Gambetta, "Procedure for systematically tuning up cross-talk in the cross-resonance gate," *Physical Review A*, vol. 93, no. 6, p. 060302, 2016.
[48] Y. Shi, N. Leung, P. Gokhale, Z. Rossi, D. I. Schuster, H. Hoffmann, and F. T. Chong, "Optimized compilation of aggregated instructions for realistic quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 1031–1044.
[49] S. S. Tannu and M. K. Qureshi, "A case for variability-aware policies for nisq-era quantum computers," *arXiv preprint arXiv:1805.10224*, 2018.
[50] R. Versluis, S. Poletto, N. Khammassi, B. Tarasinski, N. Haider, D. Michalak, A. Bruno, K. Bertels, and L. DiCarlo, "Scalable quantum circuit and control for a superconducting surface code," *Physical Review Applied*, vol. 8, no. 3, p. 034021, 2017.
[51] D. J. Welsh and M. B. Powell, "An upper bound for the chromatic number of a graph and its application to timetabling problems," *The Computer Journal*, vol. 10, no. 1, pp. 85–86, 1967.
[52] K. Wright, K. Beck, S. Debnath, J. Amini, Y. Nam, N. Grzesiak, J.-S. Chen, N. Pisenti, M. Chmielewski, C. Collins *et al.*, "Benchmarking an 11-qubit quantum computer," *Nature Communications*, vol. 10, no. 1, pp. 1–6, 2019.
[53] A. Zlokapa, S. Boixo, and D. Lidar, "Boundaries of quantum supremacy via random circuit sampling," *arXiv preprint arXiv:2005.02464*, 2020.