Timing and Resource-aware Mapping of Quantum Circuits to Superconducting Processors

Lao, Lingling; van Someren, Hans; Ashraf, Imran; Almudever, Carmen G.

DOI
10.1109/TCAD.2021.3057583

Publication date
2021

Document Version
Accepted author manuscript

Published in
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3057583, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Timing and Resource-aware Mapping of Quantum Circuits to Superconducting Processors

Lingling Lao, Hans van Someren, Imran Ashraf and Carmen G. Almudever

***Abstract*—Quantum algorithms need to be compiled to respect the constraints imposed by quantum processors, which is known as the mapping problem. The mapping procedure will result in an increase of the number of gates and of the circuit latency, decreasing the algorithm's success rate. It is crucial to minimize mapping overhead, especially for Noisy Intermediate-Scale Quantum (NISQ) processors that have relatively short qubit coherence times and high gate error rates. Most of prior mapping algorithms have only considered constraints such as the primitive gate set and qubit connectivity, but the actual gate duration and the restrictions imposed by the use of shared classical control electronics have not been taken into account. In this paper, we present a mapper called Qmap to make quantum circuits executable on scalable processors with the objective of achieving the shortest circuit latency. In particular, we propose an approach to formulate the classical control restrictions as resource constraints in a conventional list scheduler with polynomial complexity. Furthermore, we implement a routing heuristic to cope with the connectivity limitation. This router finds a set of movement operations that minimally extends circuit latency. To analyze the mapping overhead and evaluate the performance of different mappers, we map 56 quantum benchmarks onto a superconducting processor named Surface-17. Compared to a prior mapping strategy that minimizes the number of operations, Qmap can reduce the latency overhead up to 47.3% and operation overhead up to 28.6%, respectively.**

***Index Terms*—Quantum computing, quantum compilation, resource-constrained scheduling, routing**

I. Introduction

Quantum computing is entering the Noisy Intermediate-Scale Quantum (NISQ) era [1]. This refers to exploiting quantum processors consisting of only 50 to a few hundreds of noisy qubits - i.e. qubits with a relatively short coherence time and faulty operations. Due to the limited number of qubits, hardly or no quantum error correction (QEC) will be used in the next coming years, posing a limitation on the size of the quantum applications that can be successfully run on NISQ processors. Nevertheless, these processors will still be useful to explore quantum physics, and implement small quantum algorithms that will hopefully demonstrate quantum advantage [2]. For running quantum applications on NISQ devices, it is thus crucial to minimize their size in terms of circuit width (number of qubits), number of gates, and circuit latency/depth (number of cycles/steps). In addition, these quantum applications have to be adapted to the hardware constraints imposed by quantum processors. The main constraints include:

- **Primitive gate set:** Generally, only a limited set of quantum gates that can be realized with relatively high fidelity will be predefined on a quantum device. Each quantum technology may support a specific universal set of single-qubit and two-qubit gates, which are called primitive gates. Different primitive gates may have different gate durations. For instance, some superconducting quantum technologies have CZ as a primitive two-qubit gate of which the duration is twice as long as of a single-qubit primitive gate [3].
- **Qubit connectivity:** quantum technologies such as superconducting qubits [4]–[6] and quantum dots [7], [8] arrange their qubits in 1D/2D architectures with *nearest-neighbour* (NN) interactions. This means that only neighbouring qubits can interact or in other words, qubits are required to be adjacent to perform a two-qubit gate.
- **Classical control:** classical electronics are required for controlling and operating the qubits. Using a dedicated instrument per qubit is not scalable and is a very expensive approach. Therefore, shared control is required especially when building scalable quantum processors. For instance, eight qubits share one readout signal in the IBM Quantum Hummingbird processor [9] and a single Arbitrary Waveform Generator (AWG) is used for operating on a group of qubits [10], [11].

All these constraints may vary across different quantum processors, and quantum circuits normally cannot be directly executable on these devices. A mapping procedure is required to transform a hardware-agnostic quantum circuit into a constraint-compliant one that can be realized on a given device. This mapping process i) decomposes any quantum gate into the supported primitive gates; ii) performs an initial placement of qubits and finds the set of movement operations to route non-NN qubits to adjacent positions when they need to interact; and iii) schedules operations to leverage the maximum available parallelism. Moreover, minimizing mapping overhead in terms of the number of gates and circuit execution time (latency) is critical for implementing quantum algorithms on NISQ processors.

Different solutions including both exact algorithms and heuristics have been proposed to map quantum circuits onto NISQ processors. [12]–[16] propose mapping approaches for a 2D grid qubit architecture with NN interactions. Other works [6], [17]–[27] target current quantum processors from IBM and Rigetti which have irregular qubit connections. Most of prior

Lingling Lao is with the Department of Physics and Astronomy at University College London.

Lingling Lao, Hans van Someren and Carmen G. Almudever are with QuTech and the Department of Quantum and Computer Engineering at Delft University of Technology. Imran Ashraf is with Computer Engineering Department, HITEC University, Taxila, Pakistan

works [6], [12]–[24] mainly consider the qubit connectivity and the primitive gate set constraints and their strategies focus on minimizing gate overhead. They assume that any operation takes one time-step without taking the actual gate duration into account. Moreover, they do not consider the shared classical control electronics, which restricts the parallelism of some operations. This means the output circuits from previous compilation passes need to be further scheduled by another hardware-aware translation phase such as OpenPulse from IBM [11] so that quantum operations can be performed on real qubits with correct timing without violating any classical control constraint [11], [28]. Venturelli et al. [25]–[27] consider gate duration and crosstalk constraints, but their mathematical optimization formulation of gate scheduling has exponential complexity.

This paper presents a mapper called **Qmap** to make quantum circuits executable on scalable superconducting processors with shared classical electronic controls. Qmap is embedded in the OpenQL compiler [29] and its output circuit is described by an executable low-level QASM-like code with precise timing information. In order to analyze the impact of the mapping procedure, we compile 56 benchmarks taken from RevLib [30] and QLib [31] onto the Surface-17 superconducting processor [28].

The main contributions of this paper are the following:

- We provide a comprehensive analysis of the hardware constraints of the Surface-17 processor, including the supported primitive gates with corresponding duration, the processor's topology that limits the qubit connectivity, and the classical control constraints resulting from the shared control electronics among qubits that limits the parallelism of quantum operations.
- We develop a Qmap mapper embedded in the OpenQL compiler [29] to compile a quantum circuit into one that complies with all the above constraints of Surface-17. Specifically, we propose an approach to formulate the classical control limitations as resource constraints in a conventional list scheduling algorithm. Its objective is to achieve the shortest circuit latency and therefore the highest gate-level parallelism with respect to these constraints. The complexity of the developed scheduling heuristic is polynomial in terms of the number of operations and resources, which is applicable to large-scale circuits.
- For coping with the limited qubit connectivity, we present a routing strategy in Qmap to move qubits that need to interact to be adjacent. The proposed router not only finds shortest paths that use least number of operations for moving qubits (which is the routing strategy developed in prior works) but also selects a set of movement operations that will minimally extend the overall circuit latency. Compared to a prior compilation strategy, the average reduction of latency overhead and the average reduction of gate overhead when using Qmap are 22% and 3.0%, respectively.
- To enable a flexible implementation, we provide a method to encode all hardware characteristics in a configuration file that is accessed by every module of the compiler. This flexibility also allows a comparative analysis of the mapping impacts of different characteristics, giving some directions for building future quantum devices. In addition, it allows the mapper to target different processors.
- Qmap uses not only SWAP operations (3 consecutive CNOTs) for moving qubits but also MOVE operations (2 consecutive CNOTs) when possible. Compared to the mapping by only using SWAPs in prior works, the use of MOVEs helps to reduce the number of gates and the circuit latency up to 38.9% and 29% respectively.

The rest of this paper is organized as follows. We first describe all the hardware parameters that will be considered in this work in Section II. Then we introduce the proposed resource-constrained scheduling algorithm in Section III and other modules of the developed mapper such as the routing heuristic in Section IV. Afterwards, we evaluate this mapping strategy in Section V and summarize related works in Section VI. Finally, Section VII concludes the paper and discusses future work.

II. Quantum hardware constraints

In this section, the hardware constraints of the Surface-17 superconducting processor will be briefly introduced, including the primitive gates that can be directly performed, the topology of the processor which limits interactions between qubits, and the constraints caused by the classical control electronics which impose extra limitations on the parallelism of the operations.

A. Primitive gate set

In order to run any quantum circuit, a universal set of operations needs to be implemented. In superconducting quantum processors, these operations commonly are measurement, single-qubit rotations, and multi-qubit gates.

In principle, any kind of single-qubit rotation can be performed on the Surface-17 processor. However, an infinite amount of gates cannot be predefined. In this work, we will limit single qubit gates to X and Y rotations (easier to implement), and more specifically $\pm$ 45, $\pm$ 90 and $\pm$ 180 degrees will be used in our decomposition. The primitive two-qubit gate on this processor is the conditional-phase (CZ) gate. Table I shows the gate duration (gate execution time) of single-qubit gates, CZ gate and measurement (in the Z basis) [32]. After mapping, the output circuit will only contain operations that belong to this primitive gate set. The decomposition for $Z, H, S, S^\dagger, T, T^\dagger$, CNOT, SWAP and MOVE gates into these primitive gates is shown in Figure 1 (ignoring the global phase).

TABLE I: The gate duration in cycles (each cycle represents 20 nanoseconds) of the primitive gates in the Surface-17 processor.

Gate type	Duration
$R_X(\pm 45, \pm 90, \pm 180)$	1 cycle
$R_Y(\pm 45, \pm 90, \pm 180)$	1 cycle
CZ	2 cycles
M_Z	15 cycles

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3057583, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

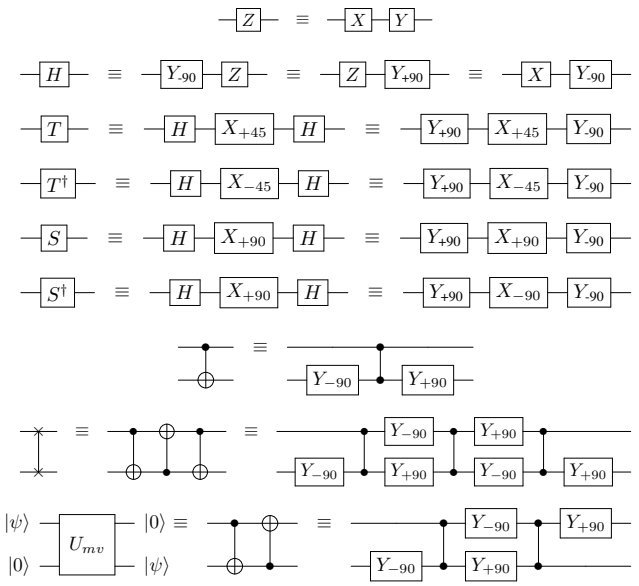


Fig. 1: Gate decomposition into primitives supported in the superconducting Surface-17 processor. U_{mv} is the MOVE operation.

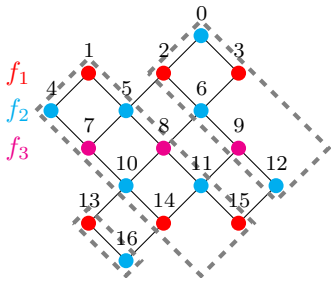


Fig. 2: Schematic of the realization of Surface-17 superconducting processor where qubits in the same color are controlled by the same frequency and $f_1 > f_1^{int} > f_2 > f_2^{park} > f_2^{int} > f_3 > f_3^{park}$.

B. Processor topology

Figure 2 shows the topology of the Surface-17 processor, where nodes represent the qubits and edges represent the connections (resonators) between them. Two-qubit gates can only be performed between connected qubits, i.e., *nearest-neighbouring* qubits. This implies that qubits that have to interact but are not placed in neighbouring positions will need to be moved to be adjacent. Quantum states in superconducting technology are usually moved using SWAP gates. A SWAP gate is implemented by three CNOTs that in the case of the Surface-17 processor need to be further decomposed into CZ and R_Y gates as shown in Figure 1. In this work, we also consider the use of a MOVE operation which only requires two CNOTs (see Figure 1). Note that a MOVE operation requires that the destination qubit where the quantum state needs to be moved to, is in the $|0\rangle$ state. As mentioned, moving qubits results in an overhead in terms of number of operations and circuit depth, which in turn will decrease the circuit reliability. Therefore, an efficient routing procedure is required to find the series of movement operations to enable all two-qubit gates with minimum overhead.

C. Classical control constraints

In principle, any qubit in a processor can be operated individually and then any combination of independent single-qubit and two-qubit operations can be performed in parallel. However, scalable quantum processors use classical control electronics with channels that are shared among several qubits. Here we will describe the constraints imposed by the classical control electronics used in the Surface-17 processor and how they affect the parallelism of quantum operations.

a) Single-qubit gates: Single-qubit gates on superconducting qubits are performed by using microwave pulses. In Surface-17, these pulses are applied at a few fixed specific frequencies to ensure scalability and precise control. The three frequencies used in Surface-17 are shown in Figure 2: single-qubit gates on red, blue and pink colored qubits are performed at frequencies f_1, f_2, and f_3, respectively [28]. In this work, we assume that same-frequency qubits are operated by the same microwave source or arbitrary waveform generator (AWG) and a vector switch matrix (VSM) is used for distributing the control pulses modulated on the waves to the corresponding qubits [10].

The consequence of sharing control electronics is that one can perform the same single-qubit gate on all or some of the qubits that share a frequency, but one cannot perform different single-qubit gates at the same time on these qubits (as these would require other pulses to be generated). For instance, an X gate can be performed simultaneously on any of the pink qubits (7, 8 and 9) but not an X and a Y operation.

b) Measurement: Measuring the qubits is done by using feedlines each of which is coupled to multiple qubits [28]. In Figure 2, qubits in the same dashed rectangle are using the same feedline, e.g., qubits 13 and 16 will be measured through the same feedline. Because measurement takes several steps in sequence, measurement of a qubit cannot start when another qubit coupled to the same feedline is being measured, but any combination of qubits that are coupled to the same feedline can be measured simultaneously at a given time. For instance, qubits 13 and 16 can be measured at time t_0, but it is not possible to start measuring qubit 13 at time t_0 and then measure qubit 16 at time t_1 if the previous measurement has not finished.

c) Two-qubit gates: As mentioned, in the processor of Figure 2 each qubit belongs to one of three frequency groups $f_1 > f_2 > f_3$, colored red, blue and pink, respectively; links between neighbouring qubits are either between qubits from f_1 and f_2, or between qubits from f_2 and f_3, i.e. between a higher frequency qubit and a next lower one. In between additional frequencies such as interaction frequency f^{int} and parking frequency f^{park} are defined and $f_1 > f_1^{int} > f_2 > f_2^{park} > f_2^{int} > f_3 > f_3^{park}$ (see the frequency arrangement and the example interactions presented in Figure 5 of [28]). Each qubit can be individually driven with one of the frequencies of its group, i.e. $\{f_i, f_i^{int}, f_i^{park}\}$.

A CZ gate between two neighbouring qubits is realized by lowering the frequency of the higher frequency qubit near

to the frequency of the lower one. For instance, a CZ gate between qubits 3 and 0 is performed by detuning qubit 3 from f_1 to f_1^{int}, which is near to the frequency f_2 of qubit 0. However, CZ gates will occur between any two neighbouring (connected) qubits which have close frequencies. For example, a CZ gate can occur between the detuned qubit 3 in f_1^{int} and its neighbour qubit 6 in f_2 in the above example. To avoid this, the qubits that should not be involved in a CZ gate must be detuned to a lower frequency. In this example, q6 needs to be detuned to its *parking frequency* f_2^{park} . Moreover, qubits in parking frequencies cannot engage in any two-qubit or single-qubit gate. In addition, when performing a CZ on qubits 3 and 0, qubit 2 must stay at f_1 (and not be detuned) to avoid interaction between qubits 2 and 0. The implementation of two-qubit gates poses limitations not only on parallelizing multiple two-qubit gates but also on the parallelism of two-qubit gates and single-qubit gates. More details can be found in [28].

Violation of these classical control constraints will cause incorrect execution of quantum operations, leading to a computational failure. Therefore, scheduling algorithms that can take these constraints into account are needed to explore the maximum available parallelism.

D. Configuration file

The hardware characteristics explained in this section are precisely described in a configuration file (in json format). It parameterizes the mapping modules that will be introduced in the next section.

a) Primitive gate set: For Surface-17, the primitive gates with all attributes including duration as listed in Table I and the gate decomposition rules corresponding to those in Figure 1 are described in full detail in the configuration file.

b) Processor topology: The topology is defined by describing each connection with its source and target qubits. In Surface-17, all edges are bidirectional, e.g., both CNOT(q_a, q_b) and CNOT(q_b, q_a) can be performed on edge $e(q_a, q_b)$. Qubits and directed qubit connections are both named by integer values taken from contiguous ranges of integer numbers starting from 0. As an example, the qubit numbering of the Surface-17 processor is shown in Figure 2; in the Surface-17 topology the number of directed qubit connections is 48.

c) Classical control constraints: For **single-qubit gates**, we use a look-up table T_{g1} to describe the available AWGs and the list of corresponding qubits that each AWG controls. Similarly for **measurement**, the feedlines (three feedlines in Surface-17) and the corresponding qubits that each feedline is coupled to are described in a look-up table T_{gm} in the configuration file. The AWGs and feedlines are both named by contiguous integer numbers starting from 0. As mentioned in Section II, it is assumed that three AWGs and three feedlines are used in Surface-17, that is, $|T_{g1}| = 3$ and $|T_{gm}| = 3$, respectively. The classical control constraints of **two-qubit gates** are defined by using two look-up tables. One called T_{g2f} describes for each connection which other connections cannot be used to execute CZ gates in parallel (24 bi-directional edges on the Surface-17 topology, i.e. $|T_{g2f}| = 48$). The other table T_{g2d} describes for each connection which set of qubits needs to be detuned in addition to one of its end-points, which means a CZ on this connection and single-qubit gates on these detuned qubits cannot be performed in parallel($|T_{g2d}| = 48$).

III. Resource-constrained scheduling

Qubits in NISQ computers have relatively short coherence times, limiting the size of circuits that can be run successfully with high fidelity. It is therefore necessary to minimize the execution time of the circuit (or $makespan$, or circuit latency) and explore the highest gate-level parallelism, which is the objective of a quantum gate scheduler. Before discussing the other mapping modules, we first introduce the proposed heuristic scheduling algorithm that can take the actual gate duration and classical control constraints into account. The circuit shown in Figure 3 will be used as an example. We refer to the qubits in the quantum circuit as virtual qubits (others call them program qubits or logical qubits). These need to be mapped to the qubits in the quantum processor called physical, real or hardware qubits or locations

A. Weighted dependency graph

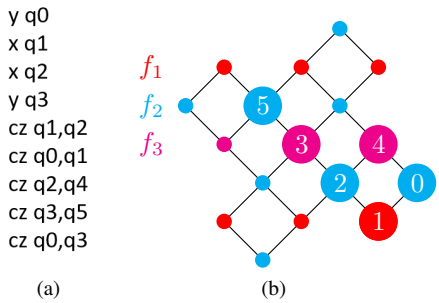
As mentioned previously, precise timing is essential for correctly executing quantum applications on real qubits. Therefore, a scheduler that considers gate duration is required to efficiently generate the correct instruction sequences with timing information meanwhile minimizing the circuit execution time. Prior works [6], [12]–[24] do not consider the actual gate duration, assuming any operation takes one time-step. To ensure quantum operations can be executed at correct time, their output circuits need to be further scheduled by some other low-level hardware-aware units such as OpenPulse [11]. In contrast, the scheduling algorithm developed in the Qmap mapper will directly take gate duration into account.

Similar to classical scheduling, a Quantum Operation Dependency Graph (QODG) $G(V_G, E_G)$ is constructed from the QASM representation of a quantum circuit, in which each operation is denoted by a node $v_i \in V_G$, and the data dependency between two operations v_i and v_j is represented by a directed edge $e(v_i, v_j) \in E_G$ with weight w_i that represents the duration of operation v_i. Pseudo source and sink nodes are added to the start and end to simplify starting and stopping iteration over the graph. The QODG of the circuit in Figure 3a is shown in Figure 4a. In previous works that do not consider gate duration, only directed graphs are constructed, which cannot be directly applied to this work.

B. Formulation of resource constraints

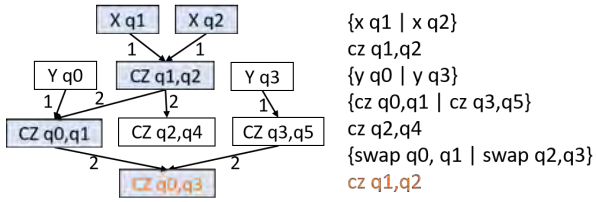
Furthermore, the scheduler also needs to adhere to the parallelism restrictions imposed by the shared classical control electronics as described in Section II. In this work, these classical control constraints are treated as **resource constraints** in an otherwise conventional critical path *list-scheduler* implementation [33]. A so-called **machine state** S is defined to describe the occupation status of each resource $r_i \in R$, where R represents the set of all resources in T_{g1}, T_{gm}, T_{g2f}, and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3057583, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



(a) (b)

Fig. 3: An example circuit consisting of 6 qubits and 9 gates. (a) Its cQASM representation without scheduling and (b) its initial qubit placement on the Surface-17 processor.



(a) (b)

Fig. 4: (a) The QODG of the circuit in Fig. 3a. Operations in the blue boxes are in the critical path. The CZ gate in orange has qubits that are not nearest neighbors. (b) The parallel cQASM code of the routed circuit, where operations in the same line or inside one bracket are scheduled to start at the same cycle. SWAP gates are inserted to perform the CZ on q0 and q3.

T_{g2d}. The constraints for *single-qubit gates* and *measurement* are implemented by using $|T_{g1}|$ and $|T_{gm}|$ resource states, respectively. To support the *two-qubit gates* constraint, there is a resource state for each connection (to constrain mutual CZ concurrency) and a resource state per qubit (to constrain CZ versus single-qubit gate concurrency). Specifically, a **resource state** consists of two elements: the operation type that is using this resource and the occupation period which is described by a pair of cycle time ($[t_0, t_1)$), representing the first cycle that it is occupied and the first cycle that it is free again, respectively. If an operation v is scheduled at cycle t_0 ($v.cycle = t_0$), then all the resources for performing v ($v.resources$) will be occupied till (and not including) $t_1 = t_0 + v.duration$ ($v.duration$ is the duration of v).

C. Scheduling heuristic

Algorithm 1 shows the pseudo code of this algorithm that schedules all gates of a given circuit with respect to the resource constraints. Its objective is to achieve the shortest circuit latency.

The heuristic maintains two sets of gates: V_m holds the gates that have been scheduled, and V_{av} includes the gates that are available for scheduling. A gate v is **available** when all its predecessors p in G have been scheduled, that is, $\forall p, p$ is in V_m. Furthermore, it maintains a machine-state S consisting of all resource states as described above.

Algorithm 1 Forward Scheduling algorithm

Input: Non-scheduled circuit
Input: Configuration file with gate durations and resource descriptions R
Output: Scheduled circuit

1: Generate QODG $G(V_G, E_G)$ from circuit
2: Initialize $\forall v \in V_G : v.resources \subset R$ and $v.duration$
3: $V_m \leftarrow$ Unique pseudo source node
4: $V_{av} \leftarrow$ All available gates in $G(V_G - V_m, E_G)$
5: Initialize cycle $t \leftarrow 0$
6: Initialize machine-state $S \leftarrow \forall r \in R$ is free
7: **while** $V_{av} \neq \varnothing$ **do**
8: $V_r \leftarrow$ resource-free gates $\subset V_{av}$ based on S
9: **if** $V_r \neq \varnothing$ **then**
10: $V_c \leftarrow$ Most-critical gates $\subset V_r$ in $G(V_G - V_m, E_G)$
11: Select $v \in V_c$ which is first in the circuit
12: Add v to V_m
13: $v.cycle \leftarrow t$
14: Update S with $v.resources$ occupied at $[t, t + v.duration)$
15: $V_{av} \leftarrow$ All available gates in $G(V_G - V_m, E_G)$
16: **else**
17: $t \leftarrow t + 1$

Algorithm 1 first constructs a QODG for the input circuit and initializes V_m, V_{av}, and S (lines 1-6). After finding all the available gates at current cycle t, it selects the ones that can be scheduled at cycle t and collects them in V_r (line 8). A gate $v \in V_{av}$ can be scheduled at cycle t only if it is **resource-free** at t, that is, when its predecessors have finished execution, $\forall p \in V_m, p.cycle + p.duration \leqslant t$ (this data dependency constraint can be seen as qubit resource constraint); and when all resources in $v.resources$ are not occupied for all cycles in $[t, t + v.duration)$. The worst-case time complexity of this step is $O(\min(g, n) \cdot (|R|))$, n and g are the number of qubits and operations in the input circuit, respectively (in the worst case, gates on every qubit can be scheduled).

If V_r is not empty, the heuristic selects the first most-critical gate v in this set (lines 9-11). A most-critical gate in V_r is the one that has the longest path to the pseudo sink node of the QODG G. In this work, the length of the longest path is pre-computed for each node in G, which only takes linear time. Then it adds this gate v to V_m, assigns the current cycle attribute to $v.cycle$. It updates S by reserving all the resources of v ($v.resources$) for its execution duration and updates V_{av} given that v has been scheduled now and thus some more gates may have become available (lines 12-15). In this case, cycle t is not incremented because more gates may be scheduled in the same cycle.

For the example circuit in Figure 4a, if X q2 is scheduled at t=0, then the resource f_2 will be occupied in $[0, 1)$ and therefore Y q0 cannot be scheduled at this cycle any more (control constraints for single-qubit gates in Section II). Furthermore, to respect with the control constraints for two-qubit gates, neither gates Y q0 and CZ q1,q2 nor gates CZ q2,q4 and CZ q3,q5 will be scheduled at the same cycle as shown in Figure 4b.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3057583, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

If V_r is empty, the heuristic increments t (line 17) and continues the schedule loop again until all the gates are scheduled, that is, V_{av} is empty. In the worst case, this loop needs to be repeated $O(L)$ times, L is the multiplication of the total number of operations (g) in the given circuit and the longest gate duration in cycles. Resource-constrained scheduling is NP-hard in the strong sense [34]. Previous works that are using exact optimization approaches or exhaustive search algorithms for scheduling [13], [18], [19], [25] cannot be adapted to efficiently solve this problem. In contrast, the proposed scheduling algorithm has reduced its complexity to at most

$$O_{schedule} = O\left(\min(g, n) \cdot (|R|) \cdot g\right). \quad (1)$$

IV. Mapping quantum algorithms

Mapping means to transform the original hardware-agnostic quantum circuit that describes the quantum algorithm to an equivalent one that can be executed on the target quantum processor. To this purpose, the mapping process has to be aware of the constraints imposed by the physical implementation of the quantum processor. These include the set of primitive gates that is supported, the allowed qubit interactions that are determined by the processor topology, and the limited concurrency of multi-gate execution because of classical control constraints. Mapping will likely increase the number of operations that are required to implement the given algorithm as well as the circuit latency/depth, decreasing the reliability of the algorithm. Efficient algorithms that can minimize this mapping overhead are then necessary, especially in NISQ processors where noise sets a limit on the maximum size of a computation that can be run successfully.

A. Overview of the Qmap mapper

The **Qmap** mapper developed in this work is embedded in the OpenQL compiler [29] and its design flow is shown in Figure 5. The input of Qmap is a quantum circuit written in OpenQL (C++ or Python). The OpenQL compiler reads and parses it to a QASM-level intermediate representation. Qmap then performs the mapping and optimization of the quantum circuit based on the processor characteristics provided in a configuration file as described in the previous section. This approach allows Qmap to target different quantum devices by just changing the parameters in the configuration file. After mapping, QASM-like code is generated. Currently, the OpenQL compiler is capable of generating cQASM [35] that can be executed on the QX simulator [36] as well as eQASM [37], a QASM-like executable code that can target the Surface-17 processor. The generation of other QASM-like languages will be part of future extensions of the OpenQL compiler. The modules of Qmap will be discussed in the rest of this section.

B. Initial placement

It is preferable to place highly interacting qubits next to each other such that less movement operations will be added for performing two-qubit gates. Similar to the placement

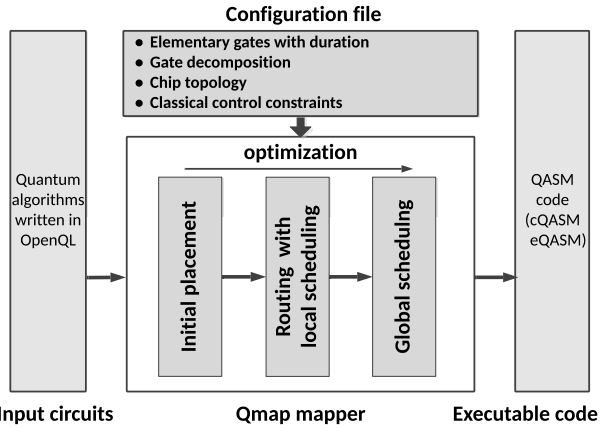


Fig. 5: Overview of the Qmap mapper embedded in the OpenQL compiler.

approaches in [38], the initial placement problem in this work is formulated as a quadratic assignment problem (QAP) and the objective is to minimize the movement or communication overhead, which is modeled by the distance between interacting qubits minus 1. Qmap tries to find an initial placement with minimum communication overhead by using the Integer Linear Programming (ILP) algorithm presented in [39]. Such an initial placement implementation can only solve small-scale problems in reasonable time. Even though for near-term implementations these numbers largely suffice, for large-scale circuits, one can either partition a large circuit into several smaller ones or apply heuristic algorithms to efficiently solve these models [40]. Other works also solve this initial placement problem by using a Satisfiability Modulo Theories (SMT) solver [41].

C. Resource-constrained routing

It is unlikely to find an initial placement in which all the qubit pairs that a two-qubit gate need to be performed on can be placed in neighboring positions. Therefore, qubits will have to be moved during computation. For instance, based on the initial placement of qubits shown in Figure 3b, the first 4 CZ gates of the circuit can be performed directly as qubits are NN, but qubits in the last CZ gate will need to be routed to adjacent positions. Routing refers to the task of finding a series of movement operations that enables the execution of two-qubit gates on a given processor topology with low communication overhead. To do so, multiple routing paths are evaluated and one is selected based on various optimization criteria such as the number of added movement operations, increase of circuit depth, or decrease of circuit reliability [6], [17]–[23], [25], [41], [42]. Afterwards, the corresponding movement operations are inserted.

1) Routing heuristic: In this work, after the ILP-based initial placement, a heuristic algorithm is used to perform this routing task. It is a scheduler-based heuristic of which the objective is to minimize overall circuit latency. Algorithm 2 shows the pseudo code of the proposed routing algorithm, which finds all two-qubit gates in which qubits are not nearest-

neighbours and inserts the required movement operations to make them adjacent. As mentioned in Section II we use SWAPs as well as MOVE operations for moving qubits.

The router algorithm starts by mapping the pseudo source node and then selecting all available gates (V_{av}) from the generated QODG (lines 1-3). Then it finds all the single-qubit gates and the two-qubit gates of which qubits are NN from V_{av}, these gates are collected in V_{nn} (line 5). If V_{nn} is not empty, then all gates in this set are mapped directly and a new set of available gates is computed (lines 6, 7, and 13-15). Mapping a (NN) gate implies replacing virtual qubit operands by their physical counterparts according to the VP-map table M similar to the one shown in Figure 3b and decomposing this gate to its primitives when the configuration specifies so.

After that, only non-NN two-qubit gates remain in the available set. The router selects the ones which are most critical in the remaining dependency graph G since they have the highest likelihood to extend the circuit when mapped in an inefficient way or when delayed (line 9). When there are several equally critical gates, the routing heuristic chooses the first one in the input circuit (line 10) and finds a set of movement operations to bring these two qubits to adjacent positions. After the movement set selection, the router schedules the SWAP/MOVE operations into the circuit (line 11), updates the VP-map (line 12), recomputes the set of available gates (line 15), and runs the routing heuristic until all the gates are mapped.

2) Movement set selection: For finding a set of movement operations for a non-NN two-qubit gate, all shortest paths between these two qubits are considered. During Qmap initialization time, the distance (i.e. the length of the shortest path) between each pair of qubits has been computed using the Floyd-Warshall algorithm. Finding all shortest paths between qubits at mapping-time is done by a breadth-first search (BFS), that is, selecting only path extensions which decrease the distance between the qubits. For each shortest path, there may exist several movement sets since qubits can meet in any neighboring position within the path. Note that all movement sets would lead to adding an **equal minimum number** of movements to the circuit. In a $\sqrt{N} \times \sqrt{N}$ grid architecture, the total number of shortest paths between most remote two nodes (q_i, q_j) is $O(4^{\sqrt{N}})$ and the number of movement sets for each path is $(2\sqrt{N} - 2)$.

In this work, a set of movement operations that **minimally extends the circuit latency** is selected and scheduled into the circuit. As shown in Algorithm 3, this router evaluates all movement sets by looking back to the previously mapped gates (lines 1 and 2) and interleaving each set of movements with those gates using the proposed resource-constrained scheduling heuristic (Section III) in an as-soon-as-possible (ASAP) policy (line 4). It selects the one(s) which minimally extend(s) the circuit latency (lines 6 and 7). When there are multiple minimal-cost sets, a random one is taken. The complexity of this routing strategy is

$$O(g\sqrt{n}4^{\sqrt{n}}) \cdot O_{schedule}. \quad (2)$$

For example, there are multiple ways to move qubits q0 and q3 in Figure 3b to be adjacent. One solution is to swap q0 with q4 and swap q2 with q3. However, these two SWAP gates cannot be performed in parallel because of the two-qubit gate control constraints in Section II. Alternatively, the router chooses the movement set {SWAP q0,q1 and SWAP q2,q3} which will minimally extend circuit latency without violating any constraints as shown in Figure 4b.

Algorithm 2 Forward Routing algorithm

Input: Non-routed circuit, VP-map M
Input: Configuration file with topology and constraints
Output: Routed circuit
1: Generate QODG $G(V_G, E_G)$
2: $V_m \leftarrow$ Unique pseudo source node
3: $V_{av} \leftarrow$ All available gates in $G(V_G - V_m, E_G)$
4: **while** $V_{av} \neq \varnothing$ **do**
5: $V_{nn} \leftarrow$ All single-qubit and NN two-qubit gates in V_{av}
6: **if** $V_{nn} \neq \varnothing$ **then**
7: Select the first most-critical gate $v \in V_{nn}$
8: **else**
9: $V_c \leftarrow$ Most-critical gates $\subset V_{av}$ in $G(V_G - V_m, E_G)$
10: Select $v \in V_c$ which is first in the circuit
11: Insert movement(s) for v
12: Update M
13: Map v according to M
14: Add v to V_m
15: $V_{av} \leftarrow$ All available gates in $G(V_G - V_m, E_G)$

Algorithm 3 Movement selection algorithm

Input: QODG $G(V_G, E_G)$, gate v, VP-map M
Input: Configuration file with topology and constraints
Output: The set of movements for v
1: $P \leftarrow$ All shortest paths for v
2: $MV_P \leftarrow$ All possible sets of movements based on P
3: **for** mv_j in MV_P **do**
4: Interleave mv_j with previous gates (looking back)
5: $L_{mv_j} \leftarrow$ circuit's latency extension by mv_j
6: **if** $L_{mv_i} = \min(\bigcup_j L_{mv_j})$ **then**
7: Select mv_i as the set of movements (randomly pick one when there are more one minimum sets)

D. Global scheduling

After routing, the circuit adheres to the processor topology constraint for two-qubit interactions and has been scheduled in an As-Soon-As-Possible (ASAP) way. The global scheduler reschedules the routed circuit to achieve the shortest circuit latency and the highest instruction-level parallelism. It does this in an As-Late-As-Possible (ALAP) way to minimize the required life-time and thus the decoherence error of each qubit. The global scheduler employs a backward version of Algorithm 1, i.e. it traverses the circuit starting from the sink node, working backwards through the circuit, decrementing t.

E. Decomposition and optimization

Starting from a quantum circuit described in cQASM format (see Figure 3), the circuit is also decomposed during mapping into one which only contains the *primitive gates* specified in the configuration file, on top of adherence to the other constraints. A circuit optimization module is also implemented to reduce the number of gates, e.g., two consecutive X gates can cancel each other out.

The decomposition and optimization can be done at every step of the mapping procedure, i.e. before, during, and after routing. Qmap reduces sequences of single qubit gates to their minimally required sequence both before and after routing. Whether decomposition is applied at a mapping step is specified in the configuration file. The implementation of the QODG represents the commutability of not only all gates with disjoint qubit operands but also the known two-qubit operations CNOT and CZ with overlapping operands, and optimizes their order during both routing and global scheduling.

The final output circuits by using the Qmap mapper are described in cQASM code with precise timing information, that is, which operations can be issued at each cycle. The output circuit can also be represented by eQASM code [37] that can be directly read by the quantum microarchitecture in [43].

V. Qmap Evaluation

In this section, we evaluate Qmap by mapping a set of benchmarks from RevLib [30] and QLib [31] on the superconducting processor Surface-17 that has a distance-3 surface-code topology [28]. All the hardware constraints discussed in Section II, including the primitive gates with their real gate duration, the topology and the electronic control constraints are taken into account. The mapping experiments are executed on a server with 2 Intel Xeon E5-2683 CPUs (56 logical cores) and 377 GB memory. The Operating System is CentOS 7.5 with Linux kernel version 3.10 and GCC version 4.8.5.

A. Benchmarks

The circuit characteristics of the used benchmarks are shown in Table II. All circuits have been decomposed into ones which only consist of gates from the universal set {Pauli, $S, S^\dagger, T, T^\dagger, H$, CNOT}. In these benchmarks, the number of qubits varies from 3 to 16, the number of gates goes from 5 to 64283, and the percentage of CNOT gates varies from 2.8% to 100%. Moreover, the minimum circuit depth and the minimum circuit latency are also included, ranging from 2 to 35572 time-steps and from 5 to 12256 cycles (using the gate duration of Surface-17 in Table I), respectively. Note that these numbers are meant to characterize the algorithms without considering the processor topology and classical control constraints.

The latter two parameters are defined as follows:

Circuit depth is the length of the circuit. It is equivalent to the total number of time-steps for executing the circuit assuming each of the gates takes one time-step.

Circuit latency refers to the execution time of the circuit considering the real gate duration. Latency and gate duration are expressed in cycles. In this paper, we assume that a cycle takes 20 nanoseconds.

In order to generate quantum circuits which are executable on real processors, extra movement operations need to be added and gate parallelism will be compromised. Other parameters after mapping these benchmarks to the Surface-17 processor are obtained, such as the number of inserted SWAP and MOVE operations and the CPU time the mapping process takes. We analyze the impact of the mapping procedure in terms of number of gates and circuit latency for Surface-17. The mapping overhead is calculated by $(X_o - X_{in})/X_{in}$, where X_{in} and X_o represent the values of the same circuit characteristic before and after mapping, respectively.

B. Prior compilation strategies

As mentioned previously, the routing algorithms in most of prior mapping works [6], [12]–[24] optimize the number of operations, that is, the number of added SWAP gates. They do not take actual gate duration and classical control limitations into account. Their output circuits need to be further scheduled by a low-level hardware unit like OpenPulse [11] such that they can be correctly executed with precise timing. In this work, we also implement such a compilation procedure called **MinPath** mapper to compare with the proposed Qmap. MinPath uses the same initial placement approach as the Qmap mapper. However, the router in MinPath randomly selects one of the movement sets along one of the shortest paths as described in Section IV-C without respecting to classical control constraints and without evaluating which set(s) will minimally extend circuit latency. The complexity of the router in MinPath is $O(g\sqrt{n}4^{\sqrt{n}})$.

Furthermore, we also introduce a **Trivial** mapper that may not be able to map the circuit with minimal latency extension but its routing strategy has linear complexity ($O(g)$). In the trivial mapping strategy, a naive initial placement is used in which qubits are just placed in their appearing order, no circuit optimization is performed. For the router in the trivial mapper, the gates in the input circuit are mapped in the order as they appear in the circuit, i.e. by-passing the QODG. For performing a non-NN two-qubit gate, it simply selects the first shortest path that is found. Moreover, only a single set of movement operations is generated for the chosen path, the set moving the control qubit adjacent to the target qubit. In addition, only SWAP gates are generated for moving qubits. By contrast, the MinPath and Qmap mappers use the ILP-based initial placement, enable circuit optimization, and can insert both SWAP and MOVE gates.

The main differences of these three mapping strategies are summarized in Table III. To provide gate sequences with precise timing and comply with the classical control constraints, the proposed resource-constrained scheduling is performed after routing procedure of the Trivial and MinPath mappers.

C. Overhead comparison of various mappers

Table IV shows the results of mapping benchmark circuits to the Surface-17 superconducting processor using three different mapping strategies: Trivial, MinPath, and Qmap. In this paper,

TABLE II: The characteristics of the input benchmarks including the number of qubits, the total number of gates, the number of two-qubit gates (CNOTs), its circuit depth and its circuit latency in cycles (20 ns per cycle).

Benchmarks	Qubits	Gates	CNOTs	Depth	Latency
alu_bdd_288	7	84	38	48	169
alu_v0_27	5	36	17	21	72
benstein_vazirani	16	35	1	5	40
4gt12_v1_89	6	228	100	130	448
4gt4_v0_72	6	258	113	137	478
4mod5_bdd_287	7	70	31	40	140
cm42a_207	14	1776	771	940	3249
cnt3_5_180	16	485	215	207	729
cuccaroAdder_1b	4	73	17	25	58
cuccaroMultiply	6	176	32	55	133
decod24_bdd_294	6	73	32	40	143
decod24_enable	6	338	149	190	669
graycode6_47	6	5	5	5	20
ham3_102	3	20	11	11	41
miller_11	3	50	23	29	105
mini_alu_167	5	288	126	162	564
mod5adder_127	6	555	239	302	1048
mod8_10_177	6	440	196	248	872
one_two_three	5	70	32	40	141
rd32_v0_66	4	34	16	18	66
rd53_311	13	275	124	124	441
rd73_140	10	230	104	92	330
rd84_142	15	343	154	110	394
sf_274	6	781	336	436	1516
shor_15	11	4792	1788	2268	7731
sqrt8_260	12	3009	1314	1659	5740
squar5_261	13	1993	869	1048	3644
sym6_145	7	3888	1701	2187	7615

Benchmarks	Qubits	Gates	CNOTs	Depth	Latency
sym9_146	12	328	148	127	450
sys6_v0_111	10	215	98	74	266
vbeAdder_2b	7	210	42	52	116
wim_266	11	986	427	514	1788
xor5_254	6	7	5	2	5
z4_268	11	3073	1343	1643	5688
adr4_197	13	3439	1498	1839	6377
9symml_195	11	34881	15232	19235	66303
clip_206	14	33827	14772	17879	61786
cm152a_212	12	1221	532	684	2366
cm85a_209	14	11414	4986	6374	21967
co14_215	15	17936	7840	8570	29608
cycle10_2_110	12	6050	2648	3384	11692
dc1_220	11	1914	833	1038	3597
dc2_222	15	9462	4131	5242	18097
dist_223	13	38046	16624	19693	68111
ham15_107	15	8763	3858	4793	16607
life_238	11	22445	9800	12511	43123
max46_240	10	27126	11844	14257	49400
mini_alu_305	10	173	77	68	242
misex1_241	15	4813	2100	2676	9240
pm1_249	14	1776	771	940	3249
radd_250	13	3213	1405	1778	6163
root_255	13	17159	7493	8835	30575
sqn_258	10	10223	4459	5458	18955
square_root_7	15	7630	3089	3830	13049
sym10_262	12	64283	28084	35572	122564
sym9_148	10	21504	9408	12087	41641

TABLE III: The main differences of the Trivial, MinPath, and Qmap mappers. n and g are the number of qubits and gates in an input circuit, respectively.

	Circuit optimization	ILP-based placement	Routing						
			Smart gate selection	Shortest path	MOVE operation	Multiple movement sets	Minimize latency	wrt. Classical controls	Complexity
Trivial	No	No	No	Yes	No	No	No	No	$O(g)$
MinPath	Yes	Yes	Yes	Yes	Yes	Yes	No	No	$O(g\sqrt{n}4^{\sqrt{n}})$
Qmap	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	$O(g\sqrt{n}4^{\sqrt{n}}) \cdot O_{schedule}$

the mapper is set to only find an ILP-based initial placement for the first ten two-qubit gates in any given circuit and computation time is limited to 10 minutes and is not included in the final CPU time. For each benchmark circuit, the mapping procedure is executed for five times and the one with minimum overhead is reported.

Compared to the circuit characteristics before mapping (Table II), no matter which strategy is applied, the mapping procedure results in high overhead for most of the benchmarks as shown in Table IV. The only exceptions are the 'benstein_v' and 'graycode6_47' circuits, because some operations in these circuits can be canceled out by the optimization module in the mapper, decreasing their circuit sizes. When the trivial mapper is used, the mapping procedure leads to a high overhead in both circuit latency and total number of gates by up to 1160% (on average 148.3%) and 800% (on average 400.1%), respectively. The MinPath mapper results in a latency overhead by up to 260% (on average 93.4%) and a gate overhead by up to 414.6% (on average 304.1%). Finally, the proposed Qmap mapper increases the circuit latency and the total number of gates by up to 260% (on average 72.1%) and 403.2% (on average 295.9%), respectively.

Furthermore, we compare the resulted overhead of these three mapping strategies as shown in Figure 6. The trivial mapper leads to the highest mapping overhead as less optimization is performed. Compared to the trivial strategy, the MinPath mapper can reduce the latency overhead and gate overhead up to 140% ('gray6_47') and 360% ('benstein_vazirani'), respectively. *The average latency (AVL) reduction and average gate (AVG) reduction are* 30% *and* 30.2%*, respectively.* Moreover, the proposed Qmap mapper has lower or equal overhead than the MinPath mapper in terms of both circuit latency and number of gates for 96.4% and 87.5% of the benchmarks, respectively. More specifically, Qmap can reduce the latency overhead up to 47.3% ('decod24_b') and decrease the gate overhead up to 28.6% ('cuccaroMultiply') compared to the MinPath mapping strategy. *The average latency (AVL) reduction and average gate (AVG) reduction are* 22% *and* 3.0%*, respectively*. This is because the router in the MinPath mapper only considers the qubit connectivity limitation and minimizes the number of operations, that is, it randomly selects a movement set that has minimum number of operations to move qubits to be neighbours. The gate duration and classical constraints will only be taken into account by a later module (such as the global scheduler in this work and the OpenPulse in IBM Qiskit [17]). In comparison, the router in Qmap uses the proposed resource-constrained scheduling approach as base and evaluates more minimum-weight movement sets to select one which minimally extends the circuit latency (Section IV).

D. Scalability and flexibility

a) Scalability: As discussed in Section IV, the complexity of the proposed resource-constrained scheduling heuristic in the worst case is still polynomial (Equation 1), making it

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3057583, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

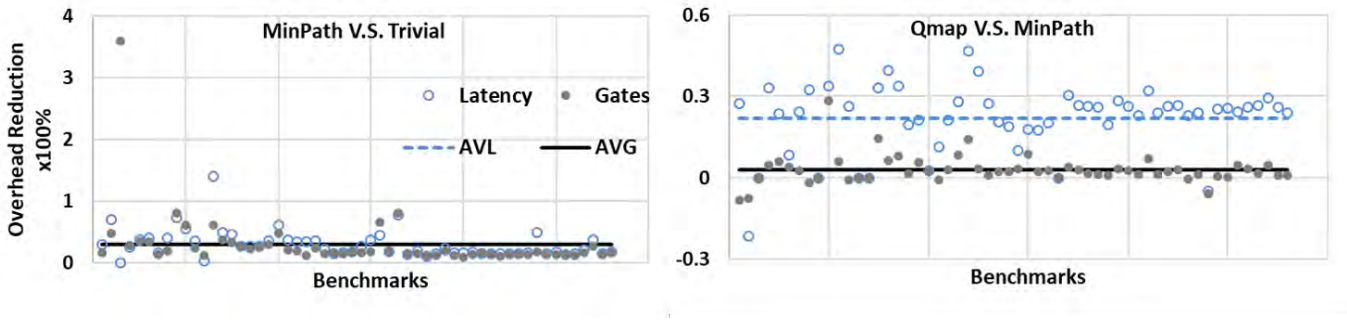


Fig. 6: Comparison of three different mapping strategies. Overhead reduction (left) when comparing the MinPath mapper to the trivial mapper and (right) when comparing Qmap to MinPath. Benchmarks are in the horizontal axis and listed in their appearing order in Table II.

applicable to large-scale quantum circuits. The complexity of the routing heuristic is polynomial in terms of the number of gates but scales sub-exponentially with the number of qubits in a given circuit when using the Qmap and MinPath strategies (in Table III).

We have tested three mapping strategies (Trivial, MinPath, Qmap) for different sizes of benchmarks, in which the number of qubits ranges from 3 to 16 and the two-qubit gate number from 5 to 62483. The runtime (in seconds) that different mappers requires for mapping each benchmark on the Surface-17 processor can be found in Table IV, which is measured by the CPU time that the entire mapping procedure takes, excluding the time the ILP-based initial placement takes. As expected, the mapper that performs more optimizations and evaluates more movement sets has a longer runtime. In this case, the trivial mapper has the shortest execution time whereas the Qmap takes the longest time. For example, when mapping the largest benchmark 'sym10_262' with 62483 gates onto the Surface-17 processor, the trivial and the Qmap mappers take 72.8 seconds and 9083.4 seconds, respectively. Moreover, most of the CPU time of MinPath and Qmap is spent on the routing procedure because of its sub-exponential complexity in qubit numbers (compared to linear complexity of the scheduling heuristic).

Based on the complexity analysis and the experimental results, we can conclude that Qmap is scalable in terms of large number of gates. However, our experiments only use benchmarks which have less 20 qubits. Therefore, its scalability with the number of qubits needs to be further investigated. Furthermore, one may need to make a compromise between mapping performance and runtime for large-scale benchmarks.

b) Flexibility: As introduced in Section II-D, the device characteristics such as the primitive gate set with gate duration, device topology, and electronic control constraints are encoded in a configuration file. Qmap will compile target quantum circuits based on the hardware information provided in this file. This means the compilation passes in Qmap including qubit initial placement, routing, scheduling, and gate decomposition are device-independent. This flexibility allows one to apply Qmap on other similar superconducting quantum processors by simply changing the corresponding device parameters in the configuration file. However, some extra changes in the compilation techniques might be required when targeting a different quantum technology, for instance, Si-spin qubits.

E. SWAPs versus MOVEs

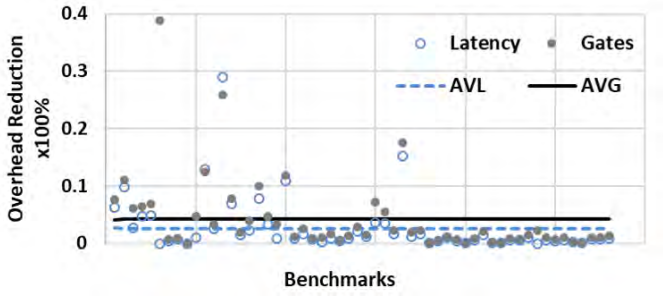


Fig. 7: Reduction of mapping overhead when using MOVEs if possible compared to when only using SWAPs. Benchmarks are in the horizontal axis and listed in their appearing order in Table II. The average latency (AVL) reduction and average gate (AVG) reduction are 2.76% and 4.21%, respectively.

As mentioned in Section II, a SWAP gate is implemented by three consecutive CNOT gates whereas a MOVE operation is implemented by two consecutive CNOT gates but requiring an ancilla qubit in the state $|0\rangle$. Therefore, if there are available ancilla qubits (qubits that are not used for computation), then it is preferable to use MOVE operations rather than SWAP gates, which helps to reduce the mapping overhead. In this section, we evaluate the benefit of using MOVE operations instead of only using SWAPs. We map the benchmarks in Table II onto the Surface-17 processor using the MinPath mapper. Different from the setups in Table IV, to have a fair comparison between using MOVEs if possible and only using SWAPs, in this case the native initial placement is applied and the first movement set is always selected. With the same qubit overhead, the mapping with MOVEs can reduce the number of gates up to 38.9% ('bestein_vazirani') and the circuit latency up to 29% ('graycode6_47') compared to the mapping with only SWAPs as shown in Figure 7. The latency reduction and gate reduction are higher than 1% for around 48.2% and 64.3% of the benchmarks, respectively.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3057583, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE IV: The results of mapping quantum benchmarks to the Surface-17 processor, including the latency overhead (LtyOH) and gate overhead (GateOH) in percentage compared to the characteristics before mapping, the number of two-qubit gates (CZs) in the mapped output circuits, the numbers of inserted SWAP (SWs) and MOVE (MVs) operations, and the CPU time that routing and scheduling take in seconds.

Benchmarks	The Trivial mapper						The MinPath mapper						The Qmap mapper					
	LtyOH	GateOH	CZs	SWs	MVs	Time	LtyOH	GateOH	CZs	SWs	MVs	Time	LtyOH	GateOH	CZs	SWS	MVs	Time
alu_bdd_288	98.2%	367.8%	113	25	0	0.06365	69.2%	306%	100	16	7	1.7313	50.3%	330.9%	109	15	13	1.77362
alu_v0_27	130.5%	422.2%	56	13	0	0.0353	38.9%	222.2%	30	3	2	4.20968	47.2%	238.9%	34	3	4	4.13529
benstein_vazirani	-10%	28.6%	10	3	0	0.01667	-10%	-74.3%	1	0	0	0.01051	-10%	-74.3%	1	0	0	0.01135
4gt12_v1_89	107.8%	422.4%	346	82	0	0.19592	81%	302.2%	270	54	4	25.6367	54%	288.6%	259	51	3	26.0342
4gt4_v0_72	135.1%	448.8%	413	100	0	0.2555	84.9%	294.6%	296	55	9	4.40794	64.8%	277.1%	273	52	2	4.628
4mod5_bdd_287	112.8%	384.3%	100	23	0	0.07120	67.1%	252.8%	71	10	5	18.7469	61.4%	242.8%	69	10	4	19.5225
cm42a_207	120.5%	394.5%	2532	587	0	1.42467	100%	344.1%	2352	517	15	611.534	75.8%	334.9%	2301	494	24	535.889
cnt3_5_180	172.3%	413.6%	725	170	0	0.38054	103%	333.6%	623	136	0	25.0301	69.5%	339.6%	641	142	0	25.6028
cuccaroAdder_1b	201.7%	134.2%	50	11	0	0.03036	55.1%	26%	23	0	3	0.2521	55.1%	26%	23	0	3	0.28906
cuccaroMultiply	213.5%	142.6%	122	30	0	0.06122	95.4%	55.7%	74	10	6	2.05933	.63.1%	39.7%	64	6	7	2.09601
decod24_bdd	120.2%	413.7%	110	26	0	0.06353	76.9%	312.3%	90	14	8	1.38109	40.6%	293.1%	83	15	3	1.46449
decod24_enable	100.6%	375.4%	467	106	0	0.23441	97.9%	233.1%	434	95	0	28.704	72%	336%	434	95	0	28.7617
graycode6_47	50%	520%	11	2	0	0.00898	-20%	200%	5	0	0	5.83973	-20%	200%	5	0	0	5.8601
ham3_102	92.7%	335%	26	5	0	0.01126	46.3%	220%	17	2	0	0.15724	46.3%	210%	17	2	0	0.22297
miller_11	89.5%	344%	65	14	0	0.02856	48.5%	232%	46	3	7	0.19096	32.3%	198%	39	0	8	0.27471
mini_alu_167	102.8%	396.9%	414	96	0	0.21319	74.6%	288.9%	309	61	0	29.411	45%	270.8%	294	56	0	28.5271
mod5adder_127	112.7%	394.4%	794	185	0	0.44677	82.1%	303.6%	645	130	8	7.0105	54.3%	279.1%	598	109	16	7.4544
mod8_10_177	108.6%	419.3%	661	155	0	0.36898	80%	310.9%	530	102	14	2.26425	64.4%	305.9%	518	106	2	2.53567
one_two_three	103.5%	394.3%	101	23	0	0.05446	66.7%	275.7%	76	12	4	6.18516	52.5%	260%	70	10	4	6.41456
rd32_v0_66	154.5%	441.2%	55	13	0	0.02766	59.9%	232.3%	32	4	2	1.65692	57.6%	226.4%	31	1	6	1.71454
rd53_311	168.2%	450.5%	448	108	0	0.24811	106.1%	354.2%	370	78	6	0.32513	94.1%	357.1%	375	81	4	0.67113
rd73_140	193.9%	420.9%	350	82	0	0.19047	127.5%	339.1%	300	62	5	20.682	100.6%1	329.6%	292	52	16	20.3441
rd84_142	251.5%	425.9%	526	124	0	0.30129	164.9%	373.4%	481	109	0	20.7494	118.5%	341.9%	448	98	0	21.1735
sf_274	121%	398.3%	1137	267	0	0.67493	78.4%	304.2%	926	178	28	40.0639	41.8%	261.3%	818	104	85	41.2879
shor_15	95%	309.2%	5472	1228	0	4.33023	74.1%	264.4%	5046	1028	87	2.45284	45.1%	255.9%	4924	982	95	14.6928
sqrt8_260	121.4%	436.1%	4719	1135	0	3.49803	102.5%	366.7%	4231	953	29	4.12037	74.5%	363.4%	4216	956	17	13.2009
squar5_261	115.8%	410.7%	2951	694	0	2.17597	97.5%	347.7%	2663	594	6	3.48788	77.5%	339.7%	2630	585	3	7.76352
sym6_145	103.1%	395.5%	5583	1294	0	4.12125	85.1%	322.5%	4872	965	138	3.94839	69%	315.2%	4787	970	88	16.7757
sym9_146	177.8%	424.7%	499	117	0	0.30173	131.1%	355.2%	447	93	10	21.1801	117.8%	343.9%	431	91	5	21.6935
sys6_v0_111	222.9%	431.2%	338	80	0	0.24816	140.6%	353.9%	290	62	3	21.1563	115.5%	322.7%	267	49	11	21.1608
vbeAdder_2b	186.2%	122.8%	135	31	0	0.09799	103.4%	42.9%	79	9	5	0.16455	85.3%	41.9%	80	6	10	0.1938
wim_266	120.4%	415.6%	1474	349	0	0.98658	98.3%	334.9%	1273	272	15	13.0377	78.4%	326.3%	1254	265	16	13.5583
xor5_254	1160%	800%	23	6	0	0.01135	260%	157.1%	8	1	0	29.7578	260%	157.1%	8	1	0	29.2384
z4_268	116.9%	413.9%	4598	1085	0	3.19178	101.5%	354.3%	4178	905	60	818.036	70.6%	340.5%	4088	887	42	869.445
adr4_197	124.1%	426.6%	5287	1263	0	3.38715	100%	361.4%	4780	1082	18	1.67818	73.6%	350.6%	4685	1021	62	10.924
9symml_195	114.3%	422.6%	53224	12664	0	36.5722	102.1%	370.8%	49485	11167	376	16.642	75.1%	364.4%	49154	11282	38	2332.7
clip_206	126.5%	432.8%	52809	12679	0	40.1273	108.1%	380.1%	49227	11379	159	17.44	80.1%	375.6%	49090	11268	257	2587.99
cm152a_212	118.3%	417.6%	1834	434	0	1.3859	90.5%	337.8%	1586	346	8	0.66896	72.7%	334.6%	1591	353	0	3.53968
cm85a_209	120.3%	434.5%	17886	4300	0	14.2237	100.8%	380.5%	16654	3832	86	6.49007	72.2%	367.5%	16224	3716	45	389.036
co14_215	156.1%	452.5%	29218	7126	0	20.9755	129.8%	414.6%	28381	6837	15	10.8777	95.8%	403.3%	27787	6615	51	1044.04
cycle10_2_110	119%	422.8%	9236	2196	0	7.12406	97.3%	365.2%	8460	1904	50	3.26144	74.9%	361.1%	8471	1899	63	106.071
dc1_220	115.2%	414.3%	2867	678	0	2.62955	97.8%	348%	2574	567	20	1.45486	66.2%	324.1%	2444	481	84	8.51783
dc2_222	118.1%	432.6%	14754	3541	0	12.5991	99.5%	371.4%	13547	3100	58	5.16826	75.7%	369%	13520	3077	79	268.637
dist_223	130%	429.4%	58891	14089	0	40.4085	111.5%	381.5%	55613	12757	359	55.0312	82.1%	372.2%	54717	12599	148	3550.7
ham15_107	118.1%	422.9%	13356	3166	0	10.1604	100.9%	364.7%	12257	2797	4	5.74947	74%	353.7%	12030	2704	30	193.48
life_238	114%	422.9%	34238	8146	0	30.3595	97.3%	365.3%	31370	7134	84	14.0716	74.9%5	366.4%	31920	7324	74	1364.49
max46_240	126.7%	421.4%	41211	9789	0	35.6086	106.1%	361.6%	37631	8375	331	16.8426	80.5%	356.7%	37565	8217	535	1840.89
mini_alu_305	216.9%	398.2%	254	59	0	0.16079	108.7%	328.3%	228	35	23	0.19804	114%	348%	242	41	21	0.40925
misex1_241	112.8%	415.1%	7206	1702	0	5.75472	96.3%	357.1%	6577	1479	20	3.13745	71.9%	354.7%	6588	1480	24	53.4152
pm1_249	120.6%	394.5%	2532	587	0	2.3615	98.4%	338.7%	2314	499	23	1.48028	73.2%	337.7%	2331	504	24	6.86838
radd_250	115.1%	419.7%	4867	1154	0	4.11447	99.4%	365.4%	4516	979	87	2.43807	75.2%	348.4%	4363	948	57	24.0061
root_255	133.2%	435.4%	26882	6463	0	20.7858	112.4%	381.4%	24991	5824	13	11.4199	83%	369.4%	24520	5627	73	844.114
sqn_258	128.6%	420.9%	15529	3690	0	11.7106	101.3%	353.6%	13908	3019	196	6.55198	74.1%	348%	13815	2984	202	270.981
square_root_7	174.1%	477.2%	12896	3269	0	10.6409	110.1%	350%	10274	2371	36	50077.7	77.8%	333.6%	9845	2184	102	46862.9
sym10_262	119.6%	429.9%	99658	23858	0	72.7567	102.1%	374.7%	92270	21030	548	42.2372	75.6%	371.5%	92326	20986	642	9083.41
sym9_148	111.1%	413.3%	32127	7573	0	27.4377	91.8%	342.8%	28378	6152	257	14.4444	69.9%	340.2%	28462	6182	254	800.226

VI. Related work

To achieve the shortest circuit latency and provide precise timing information for generating correct control signals, schedulers that consider actual gate duration should be developed. Furthermore, the control electronic constraints that can be very restrictive especially when scaling-up quantum processors, should also be taken into account to allow valid execution of quantum applications. As discussed previously, most of prior mapping works [6], [12]–[24] only focus on the primitive gate set and qubit connectivity constraints. The output circuits from prior mappers need to be further scheduled with respect with the gate duration and classical control constraints, which is less optimal than the Qmap mapper as shown in Section V. Moreover, they all use SWAP operations for moving qubits when targeting superconducting quantum processors. In addition, so far no mapper has been developed for more scalable quantum processors such as the Surface-17 processor presented in [28], [44]. Although this type of processors has been designed with the aim of building a large qubit array capable of performing fault-tolerant quantum computations based on surface code, it can be also used for running quantum algorithms in a near-term implementation.

Many existing mapping algorithms [6], [12]–[27] and this paper use either the number of inserted movement operations or the circuit depth/latency as optimization metrics. Although all these metrics affect the success probability of a quantum circuit, an analysis on which ones are more critical to be minimized is required. Recent works [20], [22], [23], [41], [42], [45] suggest to choose the routing path based on the fidelity of the two-qubit gates along the path as they are used to implement the movements (noise-aware mapper). However, the reliability of a path is calculated by simply multiplying the fidelity of each gate without considering error propagation and decoherence, which makes this metric incomplete and not

very accurate and it thus sometimes fails in predicting the most reliable route [23]. A more accurate metric that can well represent success probability and also can be easily used by the mapping procedure needs to be developed.

VII. Conclusion and Discussion

Classical control electronics will be shared among qubits for scalable quantum processors, imposing limitations on the parallelism of quantum operations. More importantly, violation of these control constraints will lead to invalid execution of quantum circuits. In this work, we have proposed a method that formulates these control constraints as resource constraints in a conventional list scheduling algorithm. Then we have developed a Qmap mapper that applies the proposed resource-constrained scheduling heuristic in the routing procedure with the objective of minimizing circuit latency. The evaluation results on the Surface-17 processor show that Qmap results in lower overhead in terms of both circuit latency and number of gates compared to the prior mapping strategy (MinPath) that minimizes the number of operations in the routing process and then reschedules the circuits with respect to the actual gate duration and classical control constraints. However, the complexity of the routing algorithm in Qmap scales sub-exponentially with the number of qubits in the input circuit. Future work can reduce its complexity by only evaluating the shortest paths where less qubits were, are or will be busy in the past, current, or coming cycles.

Furthermore, Qmap can be applied to different processors by only changing their corresponding hardware characteristics in the configuration file. We will investigate the performance of Qmap on other NISQ processors and compare it with prior works in the future. In addition, more mapping metrics need to be investigated and included in the mapper. Note that what parameter(s) to optimise during the mapping might depend on the characteristics of the target quantum processor. In addition, our mapping approach is based on the compilation of quantum circuits at the gate level. Although it generates valid instructions with precise timing, they still need to be further translated into appropriate signals that control the qubits by the microarchitecture proposed in [43]. A different approach is to directly compile quantum algorithms to control pulses [46]. Further work will compare both solutions and investigate the trade-off of allocating mapping tasks to the compiler and the microarchitecture.

Acknowledgment

The authors acknowledge support from the Intel Corporation. LLL also acknowledges funding from the China Scholarship Council.

References

[1] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[2] R. P. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, pp. 467–488, 1982.

[3] 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," *arXiv:1905.13641*, 2019.

[4] IBM, "Quantum experience," https://www.research.ibm.com/ibm-q/, 2017.

[5] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, "Characterizing quantum supremacy in near-term devices," *Nature Physics*, vol. 14, p. 595, 2018.

[6] Rigetti, "Rigetti forest," https://www.rigetti.com/forest, 2018.

[7] C. D. Hill, E. Peretz, S. J. Hile, M. G. House, M. Fuechsle, S. Rogge, M. Y. Simmons, and L. C. Hollenberg, "A surface code quantum computer in silicon," *Science advances*, vol. 1, no. 9, p. e1500707, 2015.

[8] R. Li, L. Petit, D. P. Franke, J. P. Dehollain, J. Helsen, M. Steudtner, N. K. Thomas, Z. R. Yoscovits, K. J. Singh, S. Wehner *et al.*, "A crossbar network for silicon quantum dot qubits," *Science advances*, vol. 4, no. 7, p. eaar3960, 2018.

[9] IBM, "Ibm research blog," https://www.ibm.com/blogs/research/2020/09/ibm-quantum-roadmap/, 2020.

[10] S. Asaad, C. Dickel, N. K. Langford, S. Poletto, A. Bruno, M. A. Rol, D. Deurloo, and L. DiCarlo, "Independent, extensible control of same-frequency superconducting qubits by selective broadcasting," *npj Quantum Information*, vol. 2, p. 16029, 2016.

[11] D. C. McKay, T. Alexander, L. Bello, M. J. Biercuk, L. Bishop, J. Chen, J. M. Chow, A. D. Córcoles, D. Egger, S. Filipp *et al.*, "Qiskit backend specifications for openqasm and openpulse experiments," *arXiv:1809.03452*, 2018.

[12] M. Yazdani, M. S. Zamani, and M. Sedighi, "A quantum physical design flow using ilp and graph drawing," *Quantum information processing*, vol. 12, no. 10, pp. 3239–3264, 2013.

[13] A. Lye, R. Wille, and R. Drechsler, "Determining the minimal number of swap gates for multi-dimensional nearest neighbor quantum circuits," in *The 20th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2015, pp. 178–183.

[14] R. Wille, O. Keszocze, M. Walter, P. Rohrs, A. Chattopadhyay, and R. Drechsler, "Look-ahead schemes for nearest neighbor optimization of 1d and 2d quantum circuits," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016, pp. 292–297.

[15] A. Farghadan and N. Mohammadzadeh, "Quantum circuit physical design flow for 2d nearest-neighbor architectures," *International Journal of Circuit Theory and Applications*, vol. 45, no. 7, pp. 989–1000, 2017.

[16] S. Herbert and A. Sengupta, "Using reinforcement learning to find efficient qubit routing policies for deployment in near-term quantum computers," *arXiv:1812.11619*, 2018.

[17] H. Abraham, AduOffei, I. Y. Akhalwaya, G. Aleksandrowicz *et al.*, "Qiskit: An open-source framework for quantum computing," 2019.

[18] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the ibm QX architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[19] M. Y. Siraichi, V. F. d. Santos, S. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. ACM, 2018, pp. 113–125.

[20] W. Finigan, M. Cubeddu, T. Lively, J. Flick, and P. Narang, "Qubit allocation for noisy intermediate-scale quantum computers," *arXiv:1810.08291*, 2018.

[21] 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*. ACM, 2019, pp. 1001–1014.

[22] 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 *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019, pp. 987–999.

[23] S. Nishio, Y. Pan, T. Satoh, H. Amano, and R. Van Meter, "Extracting success from ibm's 20-qubit machines using error-aware compilation," *arXiv:1903.10963*, 2019.

[24] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, "On the qubit routing problem," *arXiv:1902.08091*, 2019.

[25] D. Venturelli, M. Do, E. Rieffel, and J. Frank, "Compiling quantum circuits to realistic hardware architectures using temporal planners," *Quantum Science and Technology*, vol. 3, no. 2, p. 025004, 2018.

[26] K. E. Booth, M. Do, J. C. Beck, E. Rieffel, D. Venturelli, and J. Frank, "Comparing and integrating constraint programming and temporal planning for quantum circuit compilation," in *Twenty-Eighth International Conference on Automated Planning and Scheduling*, 2018.

[27] D. Venturelli, M. Do, B. O'Gorman, J. Frank, E. Rieffel, K. E. Booth, T. Nguyen, P. Narayan, and S. Nanda, "Quantum circuit compilation: An emerging application for automated reasoning," 2019.

[28] R. Versluis, S. Poletto, N. Khammassi, B. Tarasinski, N. Haider, D. J. Michalak, A. Bruno, K. Bertels, and L. DiCarlo, "Scalable quantum circuit and control for a superconducting surface code," *Phys. Rev. Applied*, vol. 8, no. 3, p. 034021, 2017.

[29] N. Khammassi, I. Ashraf, J. v. Someren, R. Nane, A. Krol, M. A. Rol, L. Lao, K. Bertels, and C. G. Almudever, "Openql: A portable quantum programming framework for quantum accelerators," *arXiv:2005.13283*, 2020.

[30] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "Revlib: An online resource for reversible functions and reversible circuits," in *38th International Symposium on Multiple Valued Logic (ismvl 2008)*. IEEE, 2008, pp. 220–225.

[31] C. C. Lin, A. Chakrabarti, and N. K. Jha, "QLib: Quantum module library," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 11, no. 1, p. 7, 2014.

[32] T. E. OBrien, B. Tarasinski, and L. DiCarlo, "Density-matrix simulation of small surface codes under current and projected experimental noise," *npj Quantum Information*, vol. 3, no. 1, p. 39, 2017.

[33] J. E. Kelley Jr and M. R. Walker, "Critical-path planning and scheduling," in *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*. ACM, 1959, pp. 160–173.

[34] J. Blazewicz, J. K. Lenstra, and A. R. Kan, "Scheduling subject to resource constraints: classification and complexity," *Discrete applied mathematics*, vol. 5, no. 1, pp. 11–24, 1983.

[35] N. Khammassi, G. G. Guerreschi, I. Ashraf, J. W. Hogaboam, C. G. Almudever, and K. Bertels, "cQASM v1. 0: Towards a common quantum assembly language," *arXiv:1805.09607*, 2018.

[36] N. Khammassi, I. Ashraf, X. Fu, C. G. Almudéver, and K. Bertels, "QX: A high-performance quantum computer simulation platform," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 464–469.

[37] X. Fu, L. Riesebos, M. A. rOL, J. van Straten, J. van Someren, N. Khammassi, I. Ashraf, R. Vermeulen, V. Newsum, K. Loh *et al.*, "eQASM: An executable quantum instruction set architecture," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 224–237.

[38] M. J. Dousti, A. Shafaei, and M. Pedram, "Squash: a scalable quantum mapper considering ancilla sharing," in *Proceedings of the 24th edition of the great lakes symposium on VLSI*. ACM, 2014, pp. 117–122.

[39] L. Lao, B. van Wee, I. Ashraf, J. van Someren, N. Khammassi, K. Bertels, and C. G. Almudever, "Mapping of lattice surgery-based quantum circuits on surface code architectures," *Quantum Science and Technology*, vol. 4, p. 015005, 2019.

[40] M. J. Dousti and M. Pedram, "LEQA: latency estimation for a quantum algorithm mapped to a quantum circuit fabric," in *Proceedings of the 50th Annual Design Automation Conference (DAC))*. ACM, 2013, p. 42.

[41] P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete, "Full-stack, real-system quantum computer studies: architectural comparisons and design insights," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 527–540.

[42] 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.

[43] X. Fu, M. A. Rol, C. C. Bultink, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels, "An experimental microarchitecture for a superconducting quantum processor," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50)*. IEEE/ACM, 2017, pp. 813–825.

[44] Intel, "Intel newsroom," https://newsroom.intel.com/press-kits/quantum-computing/#intel-qutech, 2019.

[45] N. M. Linke, D. Maslov, M. Roetteler, S. Debnath, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe, "Experimental comparison of two quantum computing architectures," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3305–3310, 2017.

[46] 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*. ACM, 2019, pp. 1031–1044.

Lingling Lao is a Research Fellow in quantum computing at the Department of Physics and Astronomy, University College London, UK. She received her Ph.D. at the Quantum and Computer Engineering Department and QuTech, Delft University of Technology, The Netherlands in 2019. Her research interests include quantum error correction, quantum error mitigation, and quantum compilation. Currently, she focuses on compilation and error mitigation techniques for implementing quantum applications on noisy intermediate-scale quantum computers.

Hans van Someren is a quantum software engineer at the Quantum and Computer Engineering Department and QuTech, Delft University of Technology, The Netherlands. Until 2015 he was principal architect at ACE Associated Computer Experts mainly leading the development of CoSy, a compiler generation system for advanced low-level processors such as DSPs and vector architectures, which was widely used by industry. Currently his interests are in quantum computing, i.e. compilation, scheduling, optimization and mapping, architecture exploration, programming model, and tools architecture for noisy intermediate-scale quantum computers.

Imran Ashraf is a Postdoctoral researcher at Delft University of Technology. He received his Ph.D. in Computer Engineering from Delft University of Technology, The Netherlands in 2016. The focus of his research was advanced profiling, code parallelization, communication driven mapping of applications on multicore platforms. In 2016, Imran started working as Post-Doctoral Researcher at Quantum and Computer Engineering department, QuTech, TU Delft. His research focused on compilation techniques for quantum computing. Currenty, Imran is working as Assistant Professor at Computer Engineering Department, HITEC University, Taxila, Pakistan.

Carmen G. Almudever is a group leader at the Quantum Computing Division of QuTech at Delft University of Technology. She works on the definition and implementation of a scalable quantum computer architecture that bridges the gap between quantum applications and quantum devices. Her research focuses on different aspects of the quantum computing stack including quantum programming languages and compilers, mapping of quantum algorithms, architecting and benchmarking of quantum computers, quantum error correction and fault tolerant quantum computation.