Mapping of quantum circuits onto NISQ superconducting processors

Lingling Lao,[1] Daniel M. Manzano,[2] Hans van Someren,[1] Imran Ashraf,[1] and Carmen G. Almudever[1]
[1]*Quantum Computer Architecture Lab, QuTech, Delft University of Technology, The Netherlands*
[2]*Polytechnic University of Catalonia, Spain*

A number of quantum processors consisting of a few tens of noisy qubits already exist, and are called Noisy Intermediate-Scale Quantum (NISQ) devices. Their low number of qubits precludes the use of quantum error correction procedures, and then only small-size quantum algorithms can be successfully run. These quantum algorithms need to be compiled to respect the constraints imposed by the quantum processor, known as the mapping or routing problem. The mapping will result in an increase of the number of gates and circuit depth, decreasing the algorithm's success rate.

In this paper, we present a mapper called Qmap that makes quantum circuits executable on the Surface-17 processor, a scalable processor with a surface code architecture. It takes into account not only the elementary gate set and qubit connectivity constraints but also the restrictions imposed by the use of shared classical control, which have not been considered so far. Qmap is embedded in the OpenQL compiler and uses a configuration file where the processor's characteristics are described and that makes it capable of targeting different quantum processors. To show this flexibility and evaluate its performance, we map 56 quantum benchmarks on two different superconducting quantum processors, the Surface-17 (17 qubits) and the IBM Q Tokyo (20 qubits), while using different routing strategies. We show that the best router can reduce the resulting overhead up to 80% (72%) for the number of gates and up to 71.4% (66.7%) for the circuit latency (depth) on the Surface-17 (IBM Q Tokyo) when compared to the baseline (trivial router). In addition, having a slightly higher qubit connectivity helps to decrease the number of inserted movement operations (up to 82.3%) and the use of MOVE operations instead of SWAPs can reduce the number of gates and the circuit latency up to 38.9% and 29%, respectively. Finally, we analyze how the mapping affects the reliability of some small quantum circuits. Their fidelity shows a decrease that ranges from 1.8% to 13.8%.

I. INTRODUCTION

Quantum computers promise to solve a certain set of complex problems that are intractable for even the most powerful current supercomputers, being the most famous example the factorization of large numbers using Shor's algorithm [1]. However, a fault-tolerant (FT) large-scale quantum computer with thousands or even millions of qubits will be required to solve such a kind of problem [2, 3].

Quantum computing is still far away from that as it is now just entering the Noisy Intermediate-Scale Quantum (NISQ) era [4]. 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 gates [5], [6]. 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 will 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 [4]. For running near-term quantum applications on noisy quantum devices, it is thus crucial to minimize their size in terms of circuit width (number of qubits), number of gates, and circuit depth (number of time-steps) [7, 8].

In addition, these quantum applications have to be adapted to the hardware constraints imposed by current quantum processors. The main constraints include:

- Elementary 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. For instance, some superconducting quantum technologies have CZ as an elementary two-qubit gate [9, 10].

- Qubit connectivity: quantum technologies such as superconducting qubits [5, 11–13] and quantum dots [14, 15] arrange their qubits in 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 for performing a two-qubit gate. In other technologies such as trapped-ion qubits, they are fully connected and allow all-to-all interactions [16].

- Classical control: classical electronics are required for controlling and operating the qubits. Using a dedicated instrument per qubit is not scalable and very expensive approach. Therefore, shared control is required especially when building scalable quantum processors. For instance, a single Arbitrary Waveform Generator (AWG) is used for operating on a group of qubits and several qubits are measured through the same feedline [17, 18]. This limits the possible parallelism of quantum operations, leading to longer latencies and a larger circuit depth.

All these constraints may vary between different qubit implementations and even within the same quantum technology. In order to meet them, a mapping procedure is required to transform a hardware-agnostic quantum circuit into a hardware-aware version that can be run on a given quantum processor. Mapping will: i) map virtual qubits (qubits in the circuit) to hardware qubits (physical qubits in the processor), ii) route qubits to move non-adjacent qubits to neighbouring positions when they need to interact. To this purpose, the path that the qubits will follow needs to be determined and movement operations such as shuttling in trapped ion and Si-spin quantum processors [15, 19], and SWAPs in superconducting quantum processors will be inserted accordingly. Note that routing will increase the number of operations as well as the circuit depth. iii) Schedule the operations respecting not only the dependencies between them but also the classical control constraints. In addition, gates will be decomposed to elementary gates and the circuit will be optimized at different stages of the compilation process. For NISQ processors it is key to minimize the mapping overhead such that the resulting circuit still has a high reliability and success rate. Note that the higher the number of gates and/or the circuit depth, the higher the failure rate of computation and thus the lower the reliability of the circuit.

Different solutions have been proposed to map quantum circuits onto NISQ processors. [20–28] map quantum algorithms onto processors with a 2D grid structure. [29–35] and [36, 37] propose mapping algorithms targeting IBM and Rigetti processors, respectively. Most of the works done so far focus on a specific processor architecture and mainly consider the connectivity constraint and the elementary gate set. But also other constraints of NISQ devices such as shared classical control should be taken into account to make quantum applications executable [18, 38].

These mapping algorithms usually use either the number of inserted movement operations or the circuit depth as optimization metric; that is, the routing path that inserts the least number of extra gates or the one that produces the minimal circuit depth overhead is chosen. The same metrics together with the execution time (time it takes to perform the mapping) are considered to evaluate the quality of the mapping algorithms. Although, both number of gates and circuit depth are correlated with the reliability of quantum circuits and they should be minimized as we mentioned, an analysis on how they degrade the algorithm's performance is not provided.

Recent works [8, 32, 34, 35, 39, 40], propose to use reliability as an optimization metric and analyze how the mapping process affects the success rate (also called execution success probability) of the algorithm. They 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). Note that the fidelity of two-qubit gates can vary between different pairs of qubits. However, the reliability of a path is calculated by simply multiplying the reliability of each gate without considering error propagation and decoherence, which makes this metric incomplete and not very accurate; it sometimes fails in predicting the most reliable route [35]. Based on the results presented in these papers, it seems that optimizing reliability instead of just number of gates leads to better success rates, at least for small quantum circuits.

As we showed, most of the works on mapping focus on IBM and Rigetti superconducting processors or on the UMD trapped ion processor. They only target a particular quantum processor (e.g. IBM Q Yorktown) or a family of processors (e.g. IBM Q Tenerife, IBM Q Melbourne, IBM Q Tokyo). Recently, mappers capable of generating executable circuits for different quantum processors have been presented [40, 41]. However, none of them take into account the control electronics constraints that can be very restrictive especially when scaling-up quantum processors. They do neither consider information such as gate duration (except [41]) and then assume when scheduling operations that all gates take the same number of cycles to execute. 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 [38, 42]. This processor has been designed with the aim of building a large qubit array capable of performing fault-tolerant quantum computations based on surface code. However, it can be used not only for performing QEC cycles (memory) but also for running quantum algorithms.

This paper is the first to map several quantum benchmarks to the Surface-17 processor, a scalable processor with a surface code architecture. It presents a mapper called **Qmap** that takes into account all three types of constraints: elementary gate set, qubits connectivity and control electronics. Qmap is composed of several modules, including initial placement of qubits, routing of qubits, and gate scheduling, together with decomposition and optimization steps. It is embedded in the OpenQL compiler [1] [43] that gives the flexibility to be applied to different underlying quantum processors. Hardware information such as the necessary gate decomposition rules, the processor's elementary gate set with gate duration information, processor topology, and classical control constraints are described in a configuration file that is used by different compiler passes. The mapper takes as an input a quantum program written in OpenQL (C++ or Python), maps and optimizes the corresponding quantum circuit for the given quantum platform and generates executable low-level QASM-like code [44]. The mapper is used not only for mapping quantum circuits to the

[1] OpenQL is an open-source quantum programming language and compiler developed by the Quantum Computer Architecture Lab/QuTech, Delft University of Technology. The Qmap mapper will be include in the next OpenQL release.

Surface-17 processor but also to the IBM Q Tokyo chip [5] to show its universality. Several benchmarks that differ in number of qubits and gates are evaluated. We analyze the overhead caused by mapping in terms of the number of extra gates and circuit depth/latency when using different routing strategies.

The main contributions of this paper are the following:

- We have developed a mapper (Qmap) for a scalable superconducting quantum processor such as the Surface-17. The mapper considers not only common processor constraints such as the choice of the elementary gate set and the qubit connectivity but also gate execution time (gate duration) and classical control constraints resulting from using control electronics that is shared among qubits.
- With the goal of supporting several quantum processors, our mapper has been embedded in our OpenQL compiler. It can target different quantum chips by using a configuration file in which the constraints of the processor are described. This flexibility allows performing a comparative analysis between them and give some directions for building future quantum machines. We compile 56 benchmarks taken from RevLib [45] and QLib [46] onto two quantum processors, the Surface-17 and the IBM Q Tokyo processors.
- The developed mapper supports different routing strategies. Three of them are used and evaluated in this work (trivial, base and minextendrc). After mapping (using the best router), the circuit latency (depth) can increase up to 260% (150%) and the overhead in the number of gates can be as high as 78.1% and 68% for the Surface-17 and IBM Q Tokyo, respectively.
- Our mapper uses not only SWAP operations (3 CNOTs) for moving qubits but also MOVE operations (2 CNOTs) when possible. It reduces the number of gates and the circuit latency up to 38.9% and 29%, respectively.
- An analysis on how the mapping affects the reliability of some small quantum circuits when mapped to the Surface-17 chip is also presented. They show fidelity decrease that ranges from 1.8% to 13.8%.

The rest of this paper is organized as follows. We first describes all the hardware parameters that will be considered in this work in Section II. Then we introduce the proposed mapping procedure and corresponding routing algorithms in Section III. The evaluation results are shown in Section IV. Finally, Section V concludes the paper.

II. QUANTUM HARDWARE CONSTRAINTS

In this section, the hardware constraints of the superconducting Surface-17 and the IBM Q Tokyo quantum processors 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. Elementary 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.

Surface-17 processor: 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 ± 45, ± 90 and ± 180 degrees will be used in our decomposition. The primitive two-qubit gate in this transmon 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) [47]. After mapping, the output circuit will only contain operations that belong to this elementary gate set. The decomposition for $Z, H, S, S^{\dagger}, T, T^{\dagger}$, CNOT, SWAP and MOVE gates into the elementary gates shown in Table I can be found in Appendix A.

TABLE I: The gate duration in cycles (each cycle represent 20 nanoseconds) of the elementary 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

IBM Q Tokyo (IBM-20): The elementary gates supported by the IBM Q processors are any single-qubit rotation $U(\theta, \phi, \lambda) = R_Z(\phi)R_Y(\theta)R_Z(\lambda)$ and the conditional-NOT (CNOT) gate. This means that the gate set { Pauli, $H, S, S^{\dagger}, T, T^{\dagger}$, CNOT } can be directly supported without further decomposition. In this work, we do not take the gate duration of the IBM Q Tokyo processor into account since the authors did not find the duration of two-qubit gates of this device by the time of writing this paper.

B. Processor topology

Figure 1 shows the topology of the Surface-17 and the IBM Q Tokyo processors, where nodes represent

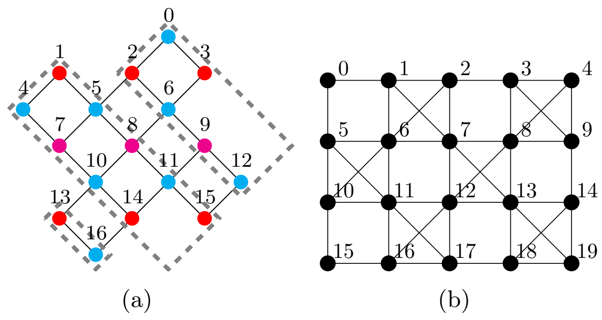


FIG. 1: (a) Schematic of the realization of the SC-17 processor and (b) the topology of the IBM Q Tokyo processor

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 6. In this work, we also consider the use of a MOVE operation which only requires two CNOTs (see Figure 6). 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.

C. Classical control constraints

In principle, any qubit in a processor can be operated individually and then any combination of 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 how the classical control electronics used in the Surface-17 processor affect the parallelism of operations. The classical control constraints for the IBM Q Tokyo processor were not found by the authors and then they will not be considered in this work.

a. Single-qubit gates: Single-qubit gates on transmons 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 1a: single-qubit gates on red, blue and pink colored qubits are performed at frequencies f_1, f_2, and f_3, respectively [38]. In this work, we assume that same-frequency qubits are operated by the same microwave source or arbitrary waveform generator and a vector switch matrix (VSM) is used for distributing the control pulses to the corresponding qubits [17].

The consequence of this 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 [38]. In Figure 1a, 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 1a 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 are defined: $f_1 > f_1^{int} > f_2 > f_2^{park} > f_2^{int} > f_3 > f_3^{park}$ (see the frequency arrangement and the example interactions presented in Figure 5 of [38]); each qubit can be individually driven with one of the frequencies of its group. 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 f_2, the frequency of qubit 0. However, CZ gates will occur between any two neighbouring qubits which have close frequencies and share a connection, e.g. between qubits 3 and 6 in the given example. To avoid this, the qubits that are not involved in the CZ gate must be kept out of the way. In this example, q6 is detuned to a lower *parking frequency*, f_2^{park} . Note that, qubits in parking frequencies cannot engage in any two-qubit or single-qubit gate. In addition, qubit 2 must stay at f_1 when qubits 3 and 0 perform a CZ, to avoid interaction between qubits 2 and 0. This example shows that the implementation of two-qubit gates poses some limitations on gate parallelism.

The hardware characteristics described in this section are included in a configuration file (in json format) that is used by all modules of the mapper.

III. MAPPING QUANTUM ALGORITHMS: THE QMAP MAPPER

Mapping means to transform the original quantum circuit that describes the quantum algorithm and is hardware-agnostic 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 elementary 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. Because of mapping, the number of operations that are required to implement the given algorithm as well as the circuit depth are likely to increase, decreasing the reliability of the algorithm. An efficient mapping is then key, 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 [43]. We show its flow in Figure 2. Its input 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 information provided in a configuration file that includes the processor topology (connectivity and number of qubits), its elementary gate set, gate decomposition rules, the duration of each gate, and the classical control constraints. After mapping, QASM-like code is generated. Currently, the OpenQL compiler is capable of generating cQASM [44] that can be executed on our QX simulator [48] as well as eQASM [49], 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.

Note that, as the characteristics of the quantum processor are described in a configuration file that is provided to the mapper, Qmap can easily target different quantum devices just by providing it with different configuration files with appropriate parameters.

The modules of Qmap will be discussed in the next sections. 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.

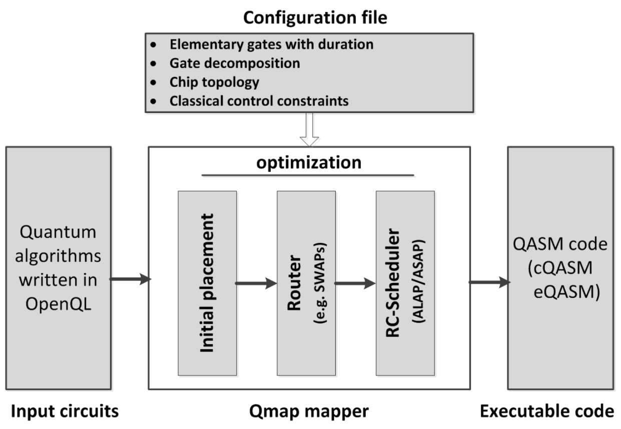
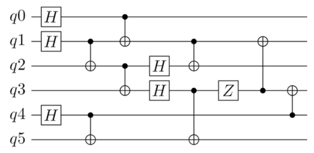


FIG. 2: Overview of the Qmap mapper embedded in the OpenQL compiler.



V	q0	q1	q2	q3	q4	q5
P	q12	q15	q11	q8	q1	q5

(a)

```
h q[0]
h q[1]
h q[4]
cnot q[1],q[2]
cnot q[4],q[5]
cnot q[0],q[1]
cnot q[2],q[3]
h q[2]
h q[3]
cnot q[1],q[2]
cnot q[3],q[5]
z q[3]
cnot q[3],q[1]
cnot q[4],q[3]
```

(b)

FIG. 3: An example circuit consisting of 6 qubits and 15 gates. (a) Its circuit description (top) and its virtual to physical qubits mapping (bottom) for the Surface-17 processor after initial placement; (b) Its cQASM representation before mapping.

B. Initial placement

Qubits are preferably placed initially such that highly interacting qubits are placed next to each other. Qmap tries to find an initial placement that minimizes the number of qubit movements by using the Integer Linear Programming (ILP) algorithm presented in [50]. Similar to the placement approaches in [24, 51, 52], the initial placement problem is formulated as a quadratic assignment problem (QAP) with the communication overhead between qubits modeled by their distance minus 1. 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 mapping models [20–22, 24, 53–55]. Other works solve this initial placement problem by using a Satisfiability Modulo Theories (SMT) solver [40].

An example circuit, its virtual to physical qubits mapping found by the initial placement module and the cQASM code before routing and scheduling are shown in Figure 3.

C. Qubit Router

It is unlikely to find an initial placement in which all interactions between the qubits are satisfied. That is, not all the qubits that will perform a two-qubit gate can be placed in neighboring positions. Therefore, they will have to be moved during computation. For instance, based on the initial placement of qubits proposed in Figure 3a, the first 6 CNOTs of the circuit can be performed directly as qubits are NN, but the last 2 CNOTs will require qubits to be routed to adjacent positions. Routing refers to the task of finding a series of movements that enables the execution of two-qubit gates on a given processor topology with low communication overhead. To do so, several routing paths are explored 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 [29–37, 39, 40]. Then, the corresponding movement operations are inserted.

In this work, next to and after the ILP-based initial placement, a heuristic algorithm is used to perform this routing task. It is a graph-based heuristic of which the objective is to achieve the shortest circuit latency and therefore the highest instruction-level parallelism. Algorithm 1 shows the pseudo code of our routing algorithm; it 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 B we use SWAPs as well as MOVE operations for moving qubits. The algorithm works as follows:

1. From the QASM representation of the quantum circuit a Quantum Operation Dependency Graph (QODG) $G(V_G, E_G)$ is constructed, 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.

2. The router algorithm starts by mapping the pseudo source node and then selecting all available operations from the input QODG, that is, the operations that do not depend on any not yet mapped operation. As long as among these are single-qubit gates or two-qubit gates with qubits that are NN, these are mapped first and a new set of available operations is computed. Mapping a (NN) gate implies replacing virtual qubit operands by their physical counterparts according to the table similar to the one shown in Figure 3a and decomposing it to its primitives when the configuration specifies so. After that, only non-NN two-qubit gates remain in the available set. The router, looking ahead to all not yet mapped operations of the circuit, selects from this availability set the ones which are most critical in the remaining dependency graph since they have the highest likelihood to extend the circuit when mapped in an inefficient way or when delayed. When there are several of these equally critical, it takes of these the first in the input circuit to map. After mapping it, it recomputes the set of available operations and runs the algorithm until there are no available operations anymore.

3. When mapping a non-NN two-qubit gate, all shortest paths between the qubits involved in this gate 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 a pair of qubits at mapping-time is done by a breadth-first search (BFS), selecting only path extensions which decrease the distance between the qubits. For each shortest path, several movement sets are computed. Each movement set consists of a sequence of movements that brings the two qubits to adjacent positions. That is, 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. To choose the best movement set, several strategies can be used that differ in how the movement set is selected and what constraints are considered. In this work, we consider three to be compared and evaluated:

   MinExtendRCRouter: As shown in Algorithm 2, this routing strategy evaluates all movement sets by looking back to the previously mapped operations and interleaving each set of movements with those operations using an as-soon-as-possible (ASAP) scheduling policy. Then, it selects the one(s) which minimally extend(s) the circuit depth or latency. When there are multiple minimal sets, a random one is taken. The scheduling in this strategy takes gate duration and the classical control resource constraints into account, the latter limiting instruction-level parallelism. Its aim is to minimize the extension of the circuit latency caused by the addition of the movements by maximizing instruction-level parallelism within the constraints of the system.

   BaseRouter: It just randomly selects one of the movement sets that are generated as described above, i.e. not evaluate them for their extension of the circuit latency or depth.

   TrivialRouter: The gates in the circuit are mapped in the order as they appear in the circuit, i.e. by-passing the QODG. Then, when there is a non-NN two-qubit gate, only the first shortest path that is found, is taken. In addition, a single movement set is generated for it; the one moving the control qubit until it is near to the target. From the movement set, only SWAPs are generated, not MOVEs.

After the movement set selection, the SWAP/MOVE operations are scheduled into the output circuit and the set of available gates and the map of virtual to physical qubits are updated.

Algorithm 1 Routing algorithm

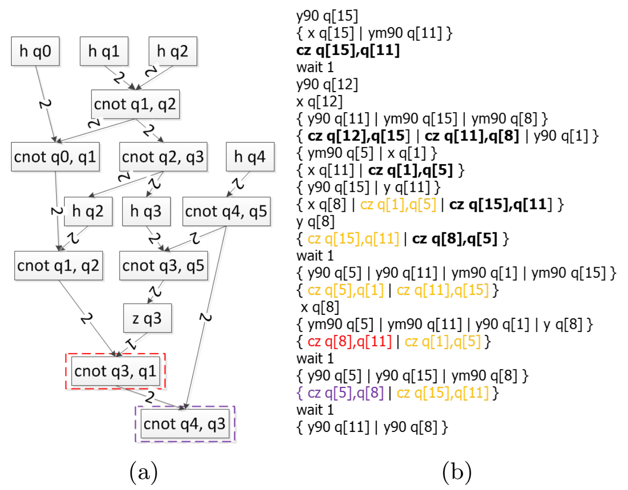
```
Input: Non-routed circuit, VP-map $M$, JSON file
Output: Routed circuit
 1: Generate QODG $G(V_G, E_G)$
 2: $V_m \leftarrow$ Unique pseudo source node
 3: $V_{av} \leftarrow$ All available gates in $G(V_G - V_m, E_G)$
 4: while $V_{av} \neq \varnothing$ do
 5:     $V_{nn} \leftarrow$ All single-qubit and NN two-qubit gates in $V_{av}$
 6:     if $V_{nn} \neq \varnothing$ then
 7:         Select $v \in V_{nn}$ arbitrarily
 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:     end if
14:     Map $v$ according to $M$
15:     Add $v$ to $V_m$
16:     $V_{av} \leftarrow$ All available gates in $G(V_G - V_m, E_G)$
17: end while
```

Algorithm 2 Movement insertion algorithm

```
Input: QODG $G(V_G, E_G)$, gate $v$, VP-map $M$, JSON file
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:     $T_{mv_j} \leftarrow$ circuit's latency extension by $mv_j$
6: end for
7: if $T_{mv_i} = min(\bigcup_j T_{mv_j})$ then
8:     Select $mv_i$ as the set of movements, picking a random
       minimum one when there are more
9: end if
```

D. RC-scheduler

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, taking the resource constraints into account only in the case of the MinExtendRC router. The RC-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, while taking the resource constraints into account. The resource constraints encode the control concurrency limitations together with the duration of the individual gates.



```
y90 q[15]
{ x q[15] | ym90 q[11] }
cz q[15],q[11]
wait 1
y90 q[12]
x q[12]
{ y90 q[11] | ym90 q[15] | ym90 q[8] }
{ cz q[12],q[15] | cz q[11],q[8] | y90 q[1] }
{ ym90 q[5] | x q[1] }
{ x q[11] | cz q[1],q[5] }
{ y90 q[15] | y q[11] }
{ x q[8] | cz q[1],q[5] | cz q[15],q[11] }
y q[8]
{ cz q[15],q[11] | cz q[8],q[5] }
wait 1
{ y90 q[5] | y90 q[11] | ym90 q[1] | ym90 q[15] }
{ cz q[5],q[1] | cz q[11],q[15] }
 x q[8]
{ ym90 q[5] | ym90 q[11] | y90 q[1] | y q[8] }
{ cz q[8],q[11] | cz q[1],q[5] }
wait 1
{ y90 q[5] | y90 q[15] | ym90 q[8] }
{ cz q[5],q[8] | cz q[15],q[11] }
wait 1
{ y90 q[11] | y90 q[8] }
```

FIG. 4: (a) The QODG of the circuit in Figure 3a. The red and purple boxed CNOTs have qubits that are not NN. (b) The cQASM code of the mapped circuit, where the CZ gates in bold are already nearest neighbouring. Movement operations (added two-qubit gates are in yellow) are inserted to perform the CZ gates in red and purple.

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 *elementary gates* specified in the **configuration file** (json file), on top of adherence to the other constraints. Next to this, it is optimized to reduce the number of operations, 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 operations to their minimally required sequence both before and after routing. The implementation of the QODG represents the commutability of all gates with disjoint qubit operands but also of the known two-qubit operations CNOT and CZ with overlapping operands, and optimizes their order, both during routing and during RC-scheduling. These optimizations are not performed in the TrivialRouter. Whether gate decomposition is to be applied at a mapping step is specified in the configuration (json) file.

The cQASM code generated after the Qmap mapper is shown in Figure 4b.

IV. QMAP EVALUATION

In this section, we evaluate the *Qmap* by mapping a set of benchmarks from RevLib [45] and QLib [46] on two superconducting processors, namely, the processor with a distance-3 surface-code topology (Surface-17) [38] and the IBM Q Tokyo (IBM-20) processor [5]. These two processors have different elementary gate sets, processor topologies, and hardware constraints as described in Section II. Specifically, for the Surface-17 processor, the elementary gates with their real gate duration, the topology and the electronic control constraints are considered. For the IBM-20 processor, only the elementary gates without considering their duration and the qubit topology are considered. All mapping experiments are executed on a sever with 2 Intel Xeon E5-2683 CPUs (56 logical cores) and 377GB memory. The Operating System is CentOS 7.5 with Linux kernel version of 3.10 and GCC version of 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 $\{\text{Pauli}, S, S^\dagger, T, T^\dagger, H, \text{CNOT}\}$. 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), respectively. Note that these numbers are meant to characterize the algorithms before being mapped to the quantum processor and therefore are obtained without considering any hardware constraint.

The latter two parameters will be also used as a metrics to evaluate our Qmap mapper. They can be 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 addition, other parameters after mapping the benchmarks to the two quantum processors are provided, such as the total number of gates and two-qubit gates, the number of inserted SWAP and MOVE operations, and the time the mapping process takes.

B. Mapping results

Table IV and Table V show the results of mapping the benchmarks to the superconducting Surface-17 processor and IBM-20 processor respectively using the three different routers: trivial, base, and minextendrc router. We take the results of the mapping with the trivial router as a baseline. In this case, a naive initial placement is used in which qubits are just placed in order, no optimization is made, and only SWAP operations are inserted. The mapping results for the base and minextendrc routers use the ILP-based initial placement. As we mentioned, this method can only solve small scale problems (small circuits) in a reasonable time. Note that the qubit initial placement is an NP-hard problem. In this paper, the mapper is set to only find an initial placement for the first ten two-qubit gates in any given circuit and computation time is limited to 10 minutes. In addition, when using the base and minextendrc routers, circuit optimizations are enabled and both SWAP and MOVE gates are inserted. The mapping procedure is executed for five times and the one with minimum overhead is reported.

1. Mapping overhead

In order to get quantum circuits which are executable on real processors, extra movement operations need to be added and gate parallelism will be compromised. We first analyze the impact of the mapping procedure in terms of number of gates, circuit latency (for Surface-17) or depth (for IBM-20) compared to the circuit characteristics before mapping in Table II. As shown in Table IV and Table V, no matter which router is applied, the mapping procedure results in high overhead for most of the benchmarks. 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.

Mapping to Surface-17: As shown in Table IV, when the trivial router is used, the mapping leads to an overhead in the circuit latency and the total number of gates ranging from 50% ('graycode6_47') to 1160% ('xor5_254') and from 122.9% ('wim_266') to 800% ('xor5_254'), respectively. The base router results in an increase of the circuit latency and the total number of gates that goes from 38.9% ('alu_v0_27') to 260% ('xor5_254')) and from 26.0% ('cuccaroAdder_1b') to 373.4% ('rd84_142')), respectively. Finally, the minextendrc router increases the circuit latency and the total number of gates from 32.4% ('miller_11') to 260% ('xor5_254')) and from 20.7% ('cuccaroAdder_1b') to 78.1% ('rd32_v0')), respectively.

Mapping to IBM-20: In Table V, it is shown that the overhead in the circuit depth and the total number of gates caused by the trivial ranges from 80.5% ('decod24_e') to 650% ('xor5_254') and from 56.5% ('cnt3_5') to 257.1% ('xor5_254'), respectively. The circuit depth after mapping with both the based and the minextendrc router has increased from 13.8% ('miller_11') to 150% ('xor5_254'). The total number of gates has increased from 10% ('ham3_102') for both routers to 72% and 68%

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

(‘rd84_142’) for the base router and the minextendrc router, respectively.

2. *Comparison of different routers*

Furthermore, we evaluate the performance of these three different routers. As expected, for both processors, the trivial router leads to the highest mapping overhead, as it is our baseline. It is also observed that in general the minextendrc router shows the best performance as it leads to the lowest increase in circuit depth/latency and number of gates (Table IV and Table V). This is because the base router includes optimizations but randomly selects one movement set. The minextendrc router optimizes circuits and evaluates more shortest movement paths to select one which minimally extends the circuit latency (Section III).

Mapping to Surface-17: As shown in Table IV, the base router always outperforms the trivial router, the latency and the number of gates can be reduced up to 71.4% (‘xor5_254’) and up to 80% (‘benstein_bazirani’), respectively. Moreover, the minextendrc router has lower or equal overhead than the base router in terms of both circuit latency and number of gates for 85.7% and 94.6% of the benchmarks, respectively. The minextendrc router can reduce the latency up to 20.5% (‘decod24_b’) and decrease the number of gates up to 10.61% (‘sf_274’) compared to the base router.

Mapping to IBM-20: Based on the mapping results in Table V, the base router can reduce the depth for 91.1% of benchmarks (up to 66.7% for ‘xor5_254’) and decrease the number of gates for 94.6% of benchmarks (up to 72% for ‘xor5_254’) compared to the trivial router. Furthermore, the minextendrc router results in a lower or equal overhead than using the base router in both circuit latency and # gates for 96.4% and 87.5% of the benchmarks, respectively. For example, the minextendrc router leads to latency reduction up to 38.2% and gate reduction up to 17.4% for the benchmark ‘4gt12_v1_89’ compared to the base router.

3. *Comparison of processor topology*

In addition, we also investigate how the processor topology affects mapping overhead in terms of the number of inserted movement operations. For a comparison between the Surface-17 and IBM-20 processors, we transform the number of movements (SWAPs and MOVEs) into the number of elementary two-qubit gates (that is, CZ for Surface-17 and CNOT for IBM-20). Based on the mapping results shown in Table IV and Table V, the IBM-20 processor requires less movement operations than the Surface-17 processor because it has more connectivity. For example, no movement operations are even needed when mapping some benchmarks ('ham3_102', 'miller_11', and 'xor5_254') to the IBM-20 processor. For other benchmarks, the IBM-20 processor can reduce the number of inserted elementary two-qubit gates up to 82.3% ('alu_v0_27') compared to the Surface-17 processor.

4. *Runtime and scalability*

We have tested the proposed mapper 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 Qmap requires for mapping each benchmark can be found in Table IV and Table V, which is measured by the CPU time that the entire mapping procedure takes (excluding the time the ILP-based initial placement takes). The router that performs more optimizations and evaluates more movement sets should have longer runtime, which is consistent with the results shown in in Table IV and Table V. The trivial router has the shortest execution time, whereas the minextendrc shows the longest one.

For example, for the largest benchmark 'sym10_262' with 62483 gates, the mapper using the trivial router only takes 72.8 seconds and 5.02 seconds for the Surface-17 processor and the IBM-20 processor, respectively. In comparison, when the minextendrc router is used, it takes 9083.4 seconds and 1698.4 seconds for the Surface-17 processor and the IBM-20 processor, respectively. Based on the above observation, we can conclude that our mapper 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. Besides, it is necessary to analyze the trade-off between mapping optimizations and runtime for large-scale benchmarks.

5. *MOVEs versus SWAPs*

As mentioned in Section II, a SWAP gate is implemented by three consecutive CNOT gates whereas a MOVE operation is implemented by two consecutive

TABLE III: The characteristics of the benchmarks before and after mapping.

Benchmark	Before mapping				After mapping			
	Qubits	Latency	Qates	CZs	Qubits	Latency	Gates	CZs
graycode6_47	6	20	5	5	6	16	15	5
xor5_254	6	5	7	5	6	18	18	8
ham3_102	3	41	20	11	3	60	62	17
cuccroadder_1b	4	58	73	17	5	90	92	23
alu_v0_27	5	72	36	17	6	100	116	30
rd32_v0_66	4	66	34	16	6	105	113	32
miller_11	3	105	50	23	4	156	166	46

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 the mapping results of Tables IV and V, MOVE operations are allowed for both base and minextendrc routers. 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 base router. Different from the setups in Table IV, to have a fair comparison between using MOVEs if possible and only using SWAPs, in this case ILP-based initial placement is not applied and the first movement set is always selected. As shown in Table VI, generating MOVEs instead of SWAPs can reduce both the number of gates up to 38.9% ('bestein_vazirani') and the circuit latency up to 29% ('graycode6_47').

6. *Fidelity analysis*

Qubits have limited coherence time and quantum operations are faulty, therefore, higher number of operations and longer circuit latency/depth will possibly lead to lower algorithm reliability which is measured by fidelity in this paper. We investigate how the mapping affects the circuit fidelity by simulating various small benchmarks on a density-matrix-based simulator called quantumsim [47]. The error models in this simulator are implemented based on experimental parameters for transmon qubits. In this work, we only consider qubit decoherence (relaxation and dephasing), gate and measurement errors, using the parameters from [47]. More specifically, the qubit relaxation time T_1 and dephasing time T_ϕ are set to be 30000 ns and 60000 ns, respectively. The in-plane error and in-axis error for single-qubit rotations are set to be $5 * 10^{-4}$ and 10^{-4}, respectively. The incoherent deviation from the expected phase value for CZ gates is $\frac{0.01}{2\pi}$and the readout error is 0.0015.

Figure 5 shows the fidelity before mapping and after mapping several small-scale benchmarks (Table III) on the Surface-17 processor. The fidelity is calculated by $f(\rho, \sigma) = \mathrm{Tr}\left(\sqrt{\rho^{1/2}\sigma\rho^{1/2}}\right)$ [56], ρ and σ are the density matrix description of quantum states. As expected, the

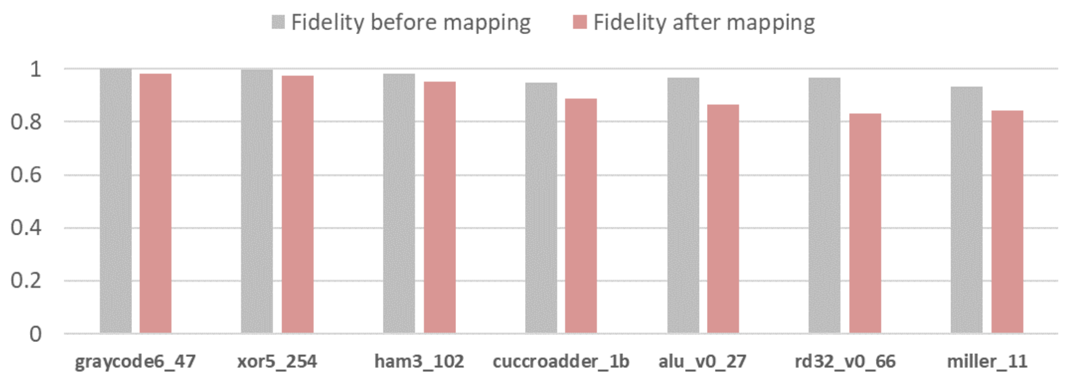


FIG. 5: The fidelity of the benchmarks before and after mapping.

fidelity of the circuits after being mapped drops. This decrease goes from 1.8% for the 'graycode6'circuit to 13.8% for 'rd32_v0_6' and it is due to insertion of more operations and the increment of the circuit latency. Moreover, for most of the benchmarks, if a benchmark has both longer latency and more gates, then it will have lower fidelity.

These two observations suggest that circuit fidelity is correlated with the latency and the number of gates. However, other parameters may also affect the fidelity such as the number of qubits and how errors propagate through two-qubit gates, and it is not clear which one has a higher impact on it. For instance, the mapped benchmarks 'miller_11' has longer latency and more gates than 'rd32_v0_6', but it achieves higher fidelity. Another example is that the mapped benchmark 'alu_v0_27' which has shorter latency but more gates achieves higher fidelity than the mapped 'rd32_v0_6'. The impact of the mapping on the algorithm fidelity needs further investigation. The next step will be then to analyze which circuit characteristics affect (most) the fidelity, and then develop a metric which not only can well represent the fidelity but also can be easily formulated to be optimized by the mapping procedure.

V. CONCLUSION AND DISCUSSION

In this work, we have presented a mapper called Qmap to make quantum circuits executable on the Surface-17 chip. It takes into account common processor constraints such as the elementary gate set and qubit connectivity, as well as classical control electronics restrictions. Qmap has been embedded in the OpenQL compiler and consists of several modules, including qubit initial placement and routing, operation scheduling, and gate decomposition and optimization. It can be applied to different processors of which hardware constraints are described in a configuration file.

We mapped 56 quantum benchmarks on two superconducting processors, which are the surface-17 processor and the IBM Q Tokyo processor. Three different routers, namely, trivial, base, and minextendrc, were used in this evaluation by the Qmap mapper. For both processors, the mapping using the minextendrc router results in the lowest overhead in terms of both circuit latency/depth and number of gates. Furthermore, as expected, the IBM-20 processor requires less movement operations compared to the Surface-17 processor due to its slightly higher qubit connectivity. We also showed that the use of a cheaper movement operation (MOVE) helps to substantially reduce the resulting overhead in terms of both added gates and latency.

We can then conclude that a flexible mapper is required for making quantum circuits executable on different real quantum processors. It must consider not only processor restrictions but also control electronic constraints as they may limit the parallelism of the operations. In addition, evaluating all possible shortest paths and different movement sets within each path and choosing one based on how well it interleaves with previous operations (look-back feature), lead to lower number of gates and circuit latency/depth. As shown, these two metrics seem to be correlated with the reliability of the algorithm, but a deeper analysis is required to develop an accurate reliability metric that can be directly used by the mapping procedure. Finally, optimizations for reducing the number of operations at different steps of the mapping process are also necessary.

Although our mapper has shown the capability to map benchmarks with large number of operations, we need to make it scalable for larger number of qubits. Future work will also include the improvement of the initial placement and routing by, for instance, finding movement operations for several two-qubit gates simultaneously. Furthermore, more mapping metrics need 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, where the generated instructions are further translated by the microarchitecture into appropriate signals that control the qubits [57]. A different approach is to directly compile quantum algorithms to control pulses [58]. Further work will compare both solutions and investigate the trade-off of allocating mapping tasks to a compiler and a microarchitecture.

ACKNOWLEDGMENTS

The authors would like to thank Xiang Fu and Adriaan Rol for enlightening discussions. We acknowledge funding from the China Scholarship Council (LL) and Intel Corporation (JS, IA, CGA).

[1] P. W. Shor, SIAM review **41**, 303 (1999).

[2] R. Van Meter and C. Horsman, Communications of the ACM **56**, 84 (2013).

[3] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, Phys. Rev. A **86**, 032324 (2012).

[4] J. Preskill, Quantum **2**, 79 (2018).

[5] IBM, "Quantum experience," (2017).

[6] 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.*, Nature **519**, 66 (2015).

[7] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, arXiv:1811.12926 (2018).

[8] N. M. Linke, D. Maslov, M. Roetteler, S. Debnath, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe, Proceedings of the National Academy of Sciences **114**, 3305 (2017).

[9] M. Kjaergaard, M. E. Schwartz, J. Braumüller, P. Krantz, J. I.-J. Wang, S. Gustavsson, and W. D. Oliver, arXiv:1905.13641 (2019).

[10] S. Caldwell, N. Didier, C. Ryan, E. Sete, A. Hudson, P. Karalekas, R. Manenti, M. da Silva, R. Sinclair, E. Acala, *et al.*, Phys. Rev. Applied **10**, 034050 (2018).

[11] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, *et al.*, Nature **508**, 500 (2014).

[12] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, Nature Physics **14**, 595 (2018).

[13] E. A. Sete, W. J. Zeng, and C. T. Rigetti, in *2016 IEEE International Conference on Rebooting Computing (ICRC)* (IEEE, 2016) pp. 1–6.

[14] C. D. Hill, E. Peretz, S. J. Hile, M. G. House, M. Fuechsle, S. Rogge, M. Y. Simmons, and L. C. Hollenberg, Science advances **1**, e1500707 (2015).

[15] 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.*, Science advances **4**, eaar3960 (2018).

[16] S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe, Nature **536**, 63 (2016).

[17] S. Asaad, C. Dickel, N. K. Langford, S. Poletto, A. Bruno, M. A. Rol, D. Deurloo, and L. DiCarlo, npj Quantum Information **2**, 16029 (2016).

[18] 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.*, arXiv:1809.03452 (2018).

[19] C. Monroe and J. Kim, Science **339**, 1164 (2013).

[20] T. S. Metodi, D. D. Thaker, A. W. Cross, F. T. Chong, and I. L. Chuang, in *Quantum Information and Computation IV*, Vol. 6244 (International Society for Optics and Photonics, 2006) p. 62440T.

[21] M. Whitney, N. Isailovic, Y. Patel, and J. Kubiatowicz, in *Proceedings of the 4th international conference on Computing frontiers* (ACM, 2007) pp. 83–94.

[22] M. J. Dousti and M. Pedram, in *DATE* (2012).

[23] M. Yazdani, M. S. Zamani, and M. Sedighi, Quantum information processing **12**, 3239 (2013).

[24] T. Bahreini and N. Mohammadzadeh, JETC **12**, 29 (2015).

[25] A. Lye, R. Wille, and R. Drechsler, in *The 20th Asia and South Pacific Design Automation Conference* (IEEE, 2015) pp. 178–183.

[26] R. Wille, O. Keszocze, M. Walter, P. Rohrs, A. Chattopadhyay, and R. Drechsler, in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)* (IEEE, 2016) pp. 292–297.

[27] A. Farghadan and N. Mohammadzadeh, International Journal of Circuit Theory and Applications **45**, 989 (2017).

[28] S. Herbert and A. Sengupta, arXiv:1812.11619 (2018).

[29] IBM, "Qiskit, quantum information software kit," (2018).

[30] A. Zulehner, A. Paler, and R. Wille, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2018).

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

[32] W. Finigan, M. Cubeddu, T. Lively, J. Flick, and P. Narang, arXiv:1810.08291 (2018).

[33] G. Li, Y. Ding, and Y. Xie, in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (ACM, 2019) pp. 1001–1014.

[34] S. S. Tannu and M. K. Qureshi, in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (ACM, 2019) pp. 987–999.

[35] S. Nishio, Y. Pan, T. Satoh, H. Amano, and R. Van Meter, arXiv:1903.10963 (2019).

[36] Rigetti, "Rigetti forest," (2018).

[37] D. Venturelli, M. Do, E. Rieffel, and J. Frank, Quantum Science and Technology **3**, 025004 (2018).

[38] R. Versluis, S. Poletto, N. Khammassi, B. Tarasinski, N. Haider, D. Michalak, A. Bruno, K. Bertels, and L. DiCarlo, Phys. Rev. Applied **8**, 034021 (2017).

[39] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (ACM, 2019) pp. 1015–1029.

[40] P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete, arXiv:1905.11349 (2019).

[41] D. Venturelli, M. Do, B. O'Gorman, J. Frank, E. Rieffel, K. E. Booth, T. Nguyen, P. Narayan, and S. Nanda, (2019).

[42] Intel, "Intel newsroom," (2019).

[43] QuTech, "Openql compiler," (2019).

[44] N. Khammassi, G. Guerreschi, I. Ashraf, J. Hogaboam, C. Almudever, and K. Bertels, arXiv:1805.09607 (2018).

[45] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, in *38th International Symposium on Multiple Valued Logic (ismvl 2008)* (IEEE, 2008) pp. 220–225.

[46] C. C. Lin, A. Chakrabarti, and N. K. Jha, ACM Journal on Emerging Technologies in Computing Systems **11**, 7 (2014).

[47] T. OBrien, B. Tarasinski, and L. DiCarlo, npj Quantum Information **3**, 39 (2017).

[48] N. Khammassi, I. Ashraf, X. Fu, C. G. Almudéver, and K. Bertels, in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017* (IEEE, 2017) pp. 464–469.

[49] X. Fu, L. Riesebos, M. Rol, J. van Straten, J. van Someren, N. Khammassi, I. Ashraf, R. Vermeulen, V. Newsum, K. Loh, *et al.*, in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (IEEE, 2019) pp. 224–237.

[50] L. Lao, B. van Wee, I. Ashraf, J. van Someren, N. Khammassi, K. Bertels, and C. Almudever, Quantum Science and Technology **4**, 015005 (2019).

[51] M. J. Dousti, A. Shafaei, and M. Pedram, in *Proceedings of the 24th edition of the great lakes symposium on VLSI* (ACM, 2014) pp. 117–122.

[52] A. Shafaei, M. Saeedi, and M. Pedram, in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)* (IEEE, 2014) pp. 495–500.

[53] S. Balensiefer, L. Kreger-Stickles, and M. Oskin, in *Quantum Information and Computation III*, Vol. 5815 (International Society for Optics and Photonics, 2005) pp. 103–114.

[54] M. J. Dousti and M. Pedram, in *Proceedings of the 50th Annual Design Automation Conference* (ACM, 2013) p. 42.

[55] M. Ahsan, *Architecture Framework for Trapped-Ion Quantum Computer based on Performance Simulation Tool*, Ph.D. thesis, Duke University (2015).

[56] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information* (Cambridge university press, 2010) p. 476.

[57] X. Fu, M. Rol, C. Bultink, J. Van Someren, N. Khammassi, I. Ashraf, R. Vermeulen, J. De Sterke, W. Vlothuizen, R. Schouten, *et al.*, in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture* (ACM, 2017) pp. 813–825.

[58] Y. Shi, N. Leung, P. Gokhale, Z. Rossi, D. I. Schuster, H. Hoffmann, and F. T. Chong, in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (ACM, 2019) pp. 1031–1044.

Appendix A: Gate decomposition

Figure 6 shows the decomposition for gates $Z, H, S, S^\dagger, T, T^\dagger$, CNOT and SWAP into the primitives of Table I.

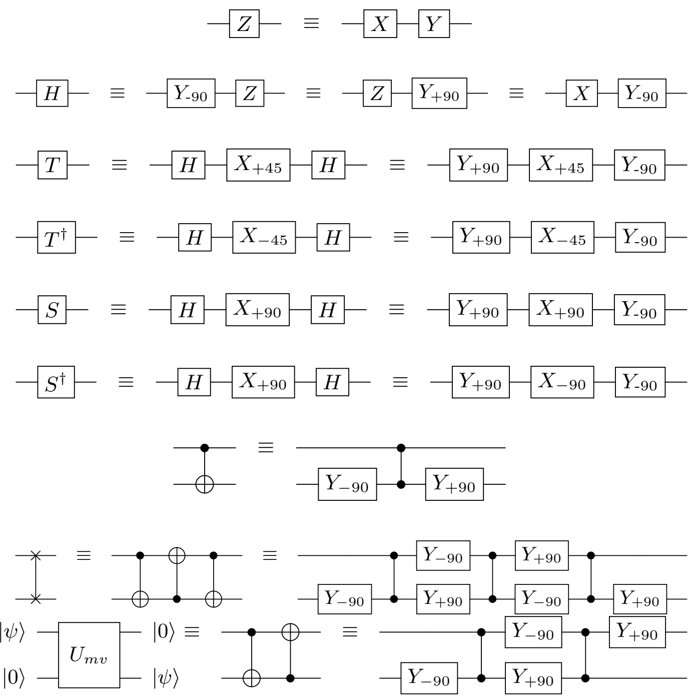


FIG. 6: Gate decomposition into primitives supported in the superconducting SC-17 processor. U_{mv} is the MOVE operation.

TABLE IV: The results of mapping quantum benchmarks to the Surface-17 processor, including the total number of gates and the number of two-qubit gates (CZs) in the mapped output circuits, the circuit latency in cycles (20 ns per cycle), the numbers of inserted SWAP (SWs) and MOVE (MVs) operations, and the CPU time that mapping takes in seconds.

Benchmarks	The trivial router						The base router						The minextendrc router					
	Latency	Gates	CZs	SWs	MVs	Time	Latency	Gates	CZs	SWs	MVs	Time	Latency	Gates	CZs	SWS	MVs	Time
alu_bdd_288	335	393	113	25	0	0.063651	286	341	100	16	7	1.7313	254	362	109	15	13	1.77362
alu_v0_27	166	188	56	13	0	0.035298	100	116	30	3	2	4.20968	106	122	34	3	4	4.13529
benstein_vazirani	36	45	10	3	0	0.016671	36	9	1	0	0	0.010512	36	9	1	0	0	0.011352
4gt12_v1_89	931	1191	346	82	0	0.195921	811	917	270	54	4	25.6367	690	886	259	51	3	26.0342
4gt4_v0_72	1124	1416	413	100	0	0.255498	884	1018	296	55	9	4.40794	788	973	273	52	2	4.628
4mod5_bdd_287	298	339	100	23	0	0.071202	234	247	71	10	5	18.7469	226	240	69	10	4	19.5225
cm42a_207	7167	8782	2532	587	0	1.42467	6499	7887	2352	517	15	611.534	5713	7724	2301	494	24	535.889
cnt3_5_180	1985	2491	725	170	0	0.38054	1480	2103	623	136	0	25.0301	1236	2132	641	142	0	25.6028
cuccaroAdder	175	171	50	11	0	0.030359	90	92	23	0	3	0.252098	90	92	23	0	3	0.28906
cuccaroMultiply	417	427	122	30	0	0.061224	260	274	74	10	6	2.05933	217	246	64	6	7	2.09601
decod24_bdd	315	375	110	26	0	0.063532	253	301	90	14	8	1.38109	201	287	83	15	3	1.46449
decod24_enable	1342	1607	467	106	0	0.234408	1324	1464	434	95	0	28.704	1151	1474	434	95	0	28.7617
graycode6_47	30	31	11	2	0	0.008976	16	15	5	0	0	5.83973	16	15	5	0	0	5.8601
ham3_102	79	87	26	5	0	0.011258	60	62	17	2	0	0.15724	60	62	17	2	0	0.22297
miller_11	199	222	65	14	0	0.028559	156	166	46	3	7	0.19096	139	149	39	0	8	0.27471
mini_alu_167	1144	1431	414	96	0	0.21319	985	1120	309	61	0	29.411	818	1068	294	56	0	28.5271
mod5adder_127	2229	2744	794	185	0	0.44677	1908	2240	645	130	8	7.0105	1618	2104	598	109	16	7.4544
mod8_10_177	1819	2285	661	155	0	0.368983	1570	1808	530	102	14	2.26425	1434	1786	518	106	2	2.53567
one_two_three	287	346	101	23	0	0.054458	235	263	76	12	4	6.18516	215	252	70	10	4	6.41456
rd32_v0_66	168	184	55	13	0	0.027656	105	113	32	4	2	1.65692	104	111	31	1	6	1.71454
rd53_311	1183	1514	448	108	0	0.248108	909	1249	370	78	6	0.325133	856	1257	375	81	4	0.671126
rd73_140	970	1198	350	82	0	0.190468	751	1010	300	62	5	20.682	662	988	292	52	16	20.3441
rd84_142	1385	1804	526	124	0	0.301286	1044	1624	481	109	0	20.7494	861	1516	448	98	0	21.1735
sf_274	3351	3892	1137	267	0	0.674928	2705	3157	926	178	28	40.0639	2151	2822	818	104	85	41.2879
shor_15	15082	19608	5472	1228	0	4.33023	13460	17464	5046	1028	87	2.45284	11217	17058	4924	982	95	14.6928
sqrt8_260	12708	16131	4719	1135	0	3.49803	11626	14041	4231	953	29	4.12037	10020	13944	4216	956	17	13.2009
squar5_261	7865	10178	2951	694	0	2.17597	7198	8922	2663	594	6	3.48788	6468	8764	2630	585	3	7.76352
sym6_145	15466	19266	5583	1294	0	4.12125	14094	16427	4872	965	138	3.94839	12873	16145	4787	970	88	16.7757
sym9_146	1250	1721	499	117	0	0.301726	1040	1493	447	93	10	21.1801	980	1456	431	91	5	21.6935
sys6_v0_111	859	1142	338	80	0	0.248159	640	976	290	62	3	21.1563	573	909	267	49	11	21.1608
vbeAdder_2b	332	468	135	31	0	0.097994	236	300	79	9	5	0.16455	215	298	80	6	10	0.1938
wim_266	3941	5084	1474	349	0	0.986583	3546	4289	1273	272	15	13.0377	3190	4203	1254	265	16	13.5583
xor5_254	63	63	23	6	0	0.011354	18	18	8	1	0	29.7578	18	18	8	1	0	29.2384
z4_268	12341	15792	4598	1085	0	3.19178	11463	13962	4178	905	60	818.036	9704	13537	4088	887	42	869.445
adr4_197	14296	18110	5287	1263	0	3.38715	12772	15868	4780	1082	18	1.67818	11070	15496	4685	1021	62	10.924
9symml_195	142144	182319	53224	12664	0	36.5722	134023	164219	49485	11167	376	16.642	116118	162001	49154	11282	38	2332.7
clip_206	139948	180243	52809	12679	0	40.1273	128597	162421	49227	11379	159	17.44	111253	160880	49090	11268	257	2587.99
cm152a_212	5166	6320	1834	434	0	1.3859	4508	5347	1586	346	8	0.668956	4086	5306	1591	353	0	3.53968
cm85a_209	48394	61007	17886	4300	0	14.2237	44110	54845	16654	3832	86	6.49007	37839	53363	16224	3716	45	389.036
co14_215	75821	99108	29218	7126	0	20.9755	68064	92308	28381	6837	15	10.8777	57968	90267	27787	6615	51	1044.04
cycle10_2_110	25607	31630	9236	2196	0	7.12406	23070	28148	8460	1904	50	3.26144	20458	27897	8471	1899	63	106.071
dc1_220	7740	9845	2867	678	0	2.62955	7116	8575	2574	567	20	1.45486	5979	8117	2444	481	84	8.51783
dc2_222	39466	50396	14754	3541	0	12.5991	36113	44864	13547	3100	58	5.16826	31796	44379	13520	3077	79	268.637
dist_223	156674	201426	58891	14089	0	40.4085	144079	183197	55613	12757	359	55.0312	124031	179639	54717	12599	148	3550.7
ham15_107	36221	45826	13356	3166	0	10.1604	33368	40721	12257	2797	4	5.74947	28906	39762	12030	2704	30	193.48
life_238	92286	117371	34238	8146	0	30.3595	85068	104447	31370	7134	84	14.0716	75462	104689	31920	7324	74	1364.49
max46_240	111978	141438	41211	9789	0	35.6086	101798	125209	37631	8375	331	16.8426	89164	123895	37565	8217	535	1840.89
mini_alu_305	767	862	254	59	0	0.160793	505	741	228	35	23	0.198036	518	775	242	41	21	0.409246
misex1_241	19670	24793	7206	1702	0	5.75472	18143	22002	6577	1479	20	3.13745	15892	21883	6588	1480	24	53.4152
pm1_249	7167	8782	2532	587	0	2.3615	6446	7793	2314	499	23	1.48028	5629	7774	2331	504	24	6.86838
radd_250	13254	16700	4867	1154	0	4.11447	12291	14955	4516	979	87	2.43807	10798	14408	4363	948	57	24.0061
root_255	71310	91873	26882	6463	0	20.7858	64948	82599	24991	5824	13	11.4199	55963	80542	24520	5627	73	844.114
sqn_258	43328	53252	15529	3690	0	11.7106	38165	46370	13908	3019	196	6.55198	33010	45801	13815	2984	202	270.981
square_root_7	35769	44042	12896	3269	0	10.6409	27419	34333	10274	2371	36	50077.7	23203	33088	9845	2184	102	46862.9
sym10_262	269200	340622	99658	23858	0	72.7567	247750	305153	92270	21030	548	42.2372	215185	303141	92326	20986	642	9083.41
sym9_148	87919	110393	32127	7573	0	27.4377	79881	95215	28378	6152	257	14.4444	70756	94656	28462	6182	254	800.226

TABLE V: The results of mapping quantum benchmarks to the IBM-20 processor, including the total number of gates and the number of two-qubit gates (CNOTs) in the mapped output circuits, the circuit depth, the numbers of inserted SWAP (SWs) and MOVE (MVs) operations, and the CPU time that mapping takes in seconds.

	The trivial router						The base router						The minextendrc router					
Benchmarks	Depth	Gates	CZs	SWs	MVs	Time	Depth	Gates	CNOTs	SWs	MVs	Time	Depth	Gates	CNOTs	SWs	MVs	Time
alu_bdd_288	104	150	104	22	0	0.006799	69	108	62	6	3	1.97957	52	99	53	1	6	2.19182
alu_v0_27	63	80	59	14	0	0.003466	26	42	21	0	2	0.695886	23	41	20	1	0	0.53724
benstein_vazirani	11	19	7	2	0	0.001361	5	9	1	0	0	0.011202	5	9	1	0	0	0.006134
4gt12_v1_89	260	413	265	55	0	0.017538	251	357	225	39	4	1.01012	155	295	163	17	6	1.08303
4gt4_v0_72	310	453	302	63	0	0.018983	265	386	239	42	0	1.31254	251	401	254	47	0	1.43037
4mod5_bdd_287	79	117	70	13	0	0.006572	72	110	69	8	7	10.5762	69	103	62	7	5	11.2424
cm42a_207	1951	2997	1914	381	0	0.132164	1885	2771	1752	325	3	1.54932	1552	2591	1572	267	0	2.04996
cnt3_5_180	453	759	449	78	0	0.031661	461	832	526	95	13	0.251479	366	787	481	84	7	0.403158
cuccaroAdder_1b	50	63	35	6	0	0.002176	45	57	29	4	0	0.006341	46	57	29	4	0	0.006140
cuccaroMultiply	114	163	95	21	0	0.008113	67	110	42	2	2	0.063068	67	110	42	2	2	0.064569
decod24_bdd_294	97	137	92	20	0	0.005843	58	99	54	4	5	0.220697	71	102	57	7	2	0.265615
decod24_enable	343	539	320	57	0	0.019654	385	528	325	52	10	0.282428	359	501	298	41	13	0.312647
graycode6_47	10	11	11	2	0	0.001782	5	5	5	0	0	132.424	5	5	5	0	0	134.398
ham3_102	26	34	23	4	0	0.001695	14	22	11	0	0	0.272413	14	22	11	0	0	0.335248
miller_11	73	95	62	13	0	0.003827	33	56	23	0	0	0.398531	33	56	23	0	0	0.394078
mini_alu_167	352	523	333	69	0	0.021998	343	469	287	47	10	0.358596	248	423	241	23	23	0.287731
mod5adder_127	581	909	557	106	0	0.038077	600	853	517	82	16	0.347671	460	783	447	52	26	0.429595
mod8_10_177	515	768	490	98	0	0.030661	422	616	350	44	11	1.38649	362	595	329	27	26	1.39496
one_two_three	111	147	101	23	0	0.006499	70	107	61	7	4	15.6825	60	96	50	4	3	15.3512
rd32_v0_66	46	61	43	9	0	0.003419	31	40	22	2	0	0.335373	31	40	22	2	0	0.261868
rd53_311	323	515	346	74	0	0.021726	238	455	286	50	6	0.226095	198	433	264	42	7	0.354412
rd73_140	272	449	299	65	0	0.017987	227	363	225	39	2	0.215851	174	365	227	41	0	0.248996
rd84_142	409	665	442	96	0	0.032003	274	590	375	73	1	0.226931	232	578	363	69	1	0.291646
sf_274	891	1303	810	158	0	0.050961	821	1168	715	121	8	0.292848	615	1014	561	55	30	0.505735
shor_15	4412	7660	4110	774	0	0.298939	3962	7143	3821	675	4	0.538582	3642	7121	3799	657	20	3.94586
sqrt8_260	3557	5362	3507	731	0	0.223606	3295	4941	3218	632	4	0.46522	2806	4823	3100	588	11	2.80964
squar5_261	2204	3369	2159	430	0	0.14577	1902	2996	1868	333	0	0.146514	1772	3056	1928	353	0	1.25905
sym6_145	4308	6442	4089	796	0	0.270245	4214	5931	3746	639	64	0.566357	3578	5746	3561	580	60	3.62793
sym9_146	355	589	385	79	0	0.025852	308	512	312	54	1	0.225797	244	513	313	51	6	0.301447
sys6_v0_111	258	392	257	53	0	0.016768	172	356	221	39	3	0.22927	121	332	197	27	9	0.366101
vbeAdder_2b	101	166	90	16	0	0.009988	89	154	82	6	11	0.08707	65	132	60	4	3	0.097052
wim_266	1050	1634	1027	200	0	0.068294	1006	1490	927	160	10	0.31237	959	1477	914	159	5	0.601171
xor5_254	15	25	23	6	0	0.001625	5	7	5	0	0	83.2281	5	7	5	0	0	86.1867
z4_268	3416	5233	3353	670	0	0.211396	3242	4844	3096	573	17	1.15126	2830	4807	3059	562	15	3.34761
adr4_197	3663	5663	3556	686	0	0.289353	3690	5471	3547	683	0	0.70171	3107	5447	3523	675	0	3.79805
9symml_195	40942	61211	39568	8112	0	2.41685	38966	58625	38575	7733	72	3.29246	32774	55634	35584	6778	9	453.364
clip_206	38219	58228	37401	7543	0	2.41549	35271	55144	35837	6973	73	4.69567	30193	54562	35255	6759	103	516.907
cm152a_212	1405	2071	1330	266	0	0.096682	1280	1849	1156	200	12	0.39819	1132	1852	1159	203	9	1.02137
cm85a_209	12801	19022	12036	2350	0	0.870193	12377	18424	11982	2258	111	1.41441	10576	18321	11879	2251	70	49.9608
co14_215	20373	31624	20488	4216	0	1.46214	18173	30707	20503	4199	33	2.11778	15606	30000	19796	3968	26	145.909
cycle10_2_110	7092	10474	6704	1352	0	0.483386	6794	9961	6447	1265	2	0.98825	5701	9757	6243	1197	2	11.4825
dc1_220	2007	3096	1919	362	0	0.151444	2032	2964	1883	344	9	0.40237	1647	2834	1753	300	10	1.6554
dc2_222	10331	15520	9687	1852	0	0.868323	10023	15292	9879	1880	54	0.99071	8527	14914	9501	1748	63	30.287
dist_223	41330	62812	39502	7626	0	2.55385	40875	63362	41536	8302	3	603.305	34997	61730	39904	7758	3	980.527
ham15_107	9919	14832	9453	1865	0	0.91102	9198	14159	9128	1740	25	1.15454	7810	13622	8591	1569	13	18.3199
life_238	26825	39551	25736	5312	0	1.94568	24642	36245	23410	4522	22	1.97507	21195	35599	22764	4310	17	132.28
max46_240	30804	47036	30210	6122	0	2.29855	28361	43463	27789	5311	6	2.19886	25289	43285	27611	5253	4	175.946
mini_alu_305	223	344	242	55	0	0.018605	142	257	157	22	7	0.26750	110	264	164	27	3	0.35919
misex1_241	5080	7440	4563	821	0	0.504028	5171	7465	4736	862	25	0.36909	4232	7271	4542	800	21	5.42279
pm1_249	1951	2997	1914	381	0	0.218987	1744	2604	1585	270	2	1.3032	1556	2586	1567	252	20	2.43371
radd_250	3651	5395	3451	682	0	0.361469	3402	5075	3279	624	1	0.59758	2975	4979	3183	592	1	3.43631
root_255	19367	29259	18779	3762	0	1.51796	18265	28528	18692	3733	0	1.87795	15814	27832	17996	3501	0	81.9624
sqn_258	11936	17901	11569	2370	0	0.836415	10989	16400	10486	2005	6	1.26047	9368	15742	9828	1781	13	26.8857
square_root_7	10066	14833	9320	2077	0	0.690437	8067	12862	7540	1463	31	600.994	6995	12629	7307	1382	36	618.468
sym10_262	74050	110071	70402	14106	0	5.02098	71921	106926	70006	13968	9	6.19955	60532	105318	68398	13434	6	1698.4
sym9_148	24017	35547	22683	4425	0	1.83779	22827	33047	20947	3841	8	2.19047	19952	32947	20847	3799	21	170.922

TABLE VI: Comparison of mapping results with and without using MOVE operations, including the total number of gates and the number of two-qubit gates (CZ) in the mapped output circuits, the circuit latency in cycles (20 ns per cycle), and the numbers of inserted SWAP and MOVE operations.

Benchmarks	MOVE operations are not used					MOVE operations are used				
	Latency	Gates	CZs	SWAPs	MOVEs	Latency	Gates	CZs	SWAPs	MOVEs
alu_bdd_288	340	375	110	24	0	324	351	104	18	6
alu_v0_27	144	159	47	10	0	137	148	44	7	3
4gt12_v1_89	822	992	286	62	0	770	916	265	41	21
4gt4_v0_72	1021	1237	362	83	0	921	1100	325	46	37
4mod5_bdd_287	289	315	94	21	0	281	296	89	16	5
benstein_vazirani	36	36	10	3	0	36	22	7	0	3
cm42a_207	6413	7778	2256	495	0	6387	7723	2242	481	14
cnt3_5_180	1523	2176	629	138	0	1512	2157	625	134	4
cuccaroAdder_1b	145	139	41	8	0	145	139	41	8	0
cuccaroMultiply	398	408	122	30	0	394	389	118	26	4
decod24_bdd	287	338	98	22	0	250	296	87	11	11
decod24_enable	1266	1442	413	88	0	1233	1393	400	75	13
graycode6_47	31	31	11	2	0	22	23	9	0	2
ham3_102	87	90	26	5	0	81	83	24	3	2
miller_11	206	212	62	13	0	203	208	61	12	1
mini_alu_167	1043	1216	348	74	0	1019	1168	336	62	12
mod5adder_127	2187	2541	737	166	0	2017	2287	670	99	67
mod8_10_177	1614	1968	568	124	0	1560	1875	544	100	24
one_two_three	230	295	83	17	0	228	285	80	14	3
rd32_v0_66	174	186	55	13	0	155	164	50	8	5
rd53_311	1064	1369	409	95	0	1055	1353	406	92	3
rd73_140	827	1114	326	74	0	814	1086	319	67	7
rd84_142	1115	1752	532	126	0	1107	1736	527	121	5
sf_274	2923	3366	975	213	0	2914	3333	967	205	8
shor_15	13498	18208	5292	1168	0	13377	17901	5214	1090	78
sqrt8_260	11292	13616	3984	890	0	11255	13532	3962	868	22
squar5_261	7081	8798	2573	568	0	7022	8685	2546	541	27
sym6_145	14751	17038	4944	1081	0	14447	16552	4808	945	136
sym9_146	1085	1557	460	104	0	1073	1533	454	98	6
sys6_v0_111	664	1052	314	72	0	640	977	294	52	20
vbeAdder_2b	344	475	144	34	0	332	449	138	28	6
wim_266	3605	4377	1276	283	0	3548	4282	1250	257	26
xor5_254	59	63	23	6	0	50	52	20	3	3
z4_268	11199	13812	4049	902	0	11069	13554	3979	832	70
adr4_197	12599	16008	4702	1068	0	12401	15640	4603	969	99
9symml_195	130111	156596	45649	10139	0	129997	156383	45587	10077	62
clip_206	126884	156543	46107	10445	0	126315	155589	45843	10181	264
cm152a_212	4406	5364	1555	341	0	4361	5303	1538	324	17
cm85a_209	42879	51691	15120	3378	0	42681	51300	15008	3266	112
co14_215	66055	87121	26026	6062	0	66047	87102	26022	6058	4
cycle10_2_110	22993	27807	8144	1832	0	22873	27573	8082	1770	62
dc1_220	7147	8665	2534	567	0	7040	8484	2484	517	50
dc2_222	35799	43119	12660	2843	0	35746	43042	12640	2823	20
dist_223	139285	174072	51133	11503	0	139228	173943	51100	11470	33
ham15_107	32885	39831	11658	2600	0	32699	39486	11562	2504	96
life_238	83192	100627	29378	6526	0	82711	99879	29170	6318	208
max46_240	99680	122065	35649	7935	0	98653	120295	35145	7431	504
mini_alu_305	537	804	236	53	0	537	786	232	49	4
misex1_241	17982	21417	6234	1378	0	17871	21191	6172	1316	62
pm1_249	6413	7778	2256	495	0	6387	7723	2242	481	14
radd_250	12103	14626	4285	960	0	12030	14488	4247	922	38
root_255	63744	79324	23414	5307	0	63640	79130	23361	5254	53
sqn_258	37976	45258	13153	2898	0	37964	45244	13150	2895	3
square_root_7	31238	37388	11393	2768	0	31002	37017	11305	2680	88
sym10_262	241971	293917	86188	19368	0	240301	290933	85358	18538	830
sym9_148	79993	95059	27477	6023	0	79257	93815	27137	5683	340