SpinQ: Compilation Strategies for Scalable Spin-Qubit Architectures

Paraskevopoulos, N.; Sebastiano, F.; Almudever, Carmen G.; Feld, S.

DOI
10.1145/3624484

Publication date
2023

Document Version
Final published version

Published in
ACM Transactions on Quantum Computing

SpinQ: Compilation Strategies for Scalable Spin-Qubit Architectures

NIKIFOROS PARASKEVOPOULOS and FABIO SEBASTIANO, Quantum and Computer Engineering Department, Delft University of Technology, The Netherlands and QuTech, The Netherlands
CARMEN G. ALMUDEVER, Computer Engineering Department, Universitat Politècnica de València, Spain
SEBASTIAN FELD, Quantum and Computer Engineering Department, Delft University of Technology, The Netherlands and QuTech, The Netherlands

Despite Noisy Intermediate-Scale Quantum devices being severely constrained, hardware- and algorithm-aware quantum circuit mapping techniques have been developed to enable successful algorithm executions. Not so much attention has been paid to mapping and compilation implementations for spin-qubit quantum processors due to the scarce availability of experimental devices and their small sizes. However, based on their high scalability potential and their rapid progress it is timely to start exploring solutions on such devices. In this work, we discuss the unique mapping challenges of a scalable crossbar architecture with shared control and introduce *SpinQ*, the first native compilation framework for scalable spin-qubit architectures. At the core of *SpinQ* is the *Integrated Strategy* that addresses the unique operational constraints of the crossbar while considering compilation scalability and obtaining a *O(n)* computational complexity. To evaluate the performance of *SpinQ* on this novel architecture, we compiled a broad set of well-defined quantum circuits and performed an in-depth analysis based on multiple metrics such as gate overhead, depth overhead, and estimated success probability, which in turn allowed us to create unique mapping and architectural insights. Finally, we propose novel mapping techniques that could increase algorithm success rates on this architecture and potentially inspire further research on quantum circuit mapping for other scalable spin-qubit architectures.

CCS Concepts: • **Computer systems organization** → **Quantum computing;** • **Software and its engineering** → *Compilers;* • **General and reference** → *Evaluation*; Performance;

Additional Key Words and Phrases: Quantum computing, scalable spin-qubit architectures, quantum circuit mapping challenges, compilation strategies

This work is part of the research program OTP with project number 16278, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO). This work has also been partially supported by the Spanish Ministerio de Ciencia e Innovación, European ERDF under grant PID2021-123627OB-C51 (CGA). We thank Menno Veldhorst and Hans van Someren for their fruitful discussions.
Authors' addresses: N. Paraskevopoulos, F. Sebastiano, and S. Feld, Quantum and Computer Engineering Department, Delft University of Technology, Mekelweg 4, Delft, The Netherlands, 2628 CD and QuTech, Lorentzweg 1, Delft, The Netherlands, 2628 CJ; e-mails: {N.Paraskevopoulos, F.Sebastiano, S.Feld}@tudelft.nl; C. G. Almudever, Computer Engineering Department, Universitat Politècnica de València, Camino de Vera s/n, 46022, València, Spain; e-mail: cargara2@disca.upv.es.

2643-6817/2023/12-ART4 $15.00
https://doi.org/10.1145/3624484

ACM Reference format:
Nikiforos Paraskevopoulos, Fabio Sebastiano, Carmen G. Almudever, and Sebastian Feld. 2023. SpinQ: Compilation Strategies for Scalable Spin-Qubit Architectures. *ACM Trans. Quantum Comput.* 5, 1, Article 4 (December 2023), 36 pages.
https://doi.org/10.1145/3624484

1 INTRODUCTION

The prospect of quantum computing advantage is steadily becoming a reality [2, 24, 35]. The community is anticipating further advances that will allow quantum computing systems to become practical and to reach computational advantage [9]. With such advancements, quantum computing systems are expected to solve a plethora of classically intractable problems. Until then, current quantum systems belong to the so-called **Noisy Intermediate-Scale Quantum (NISQ)** era [48], in which devices can only handle small-sized quantum circuits. This is due to limitations in increasing the number of qubits and high operational errors with the latter causing rapid quantum information deterioration. Combined with more hardware constraints, such as cross-talk and limited classical-control resources [1, 27], successful quantum circuit execution is a difficult feat. Scientists, both in academia and industry, face major engineering challenges in building both hardware and corresponding system software.

During the NISQ era, there have been significant efforts [4, 28, 29, 39, 41, 42, 47, 49, 54, 55, 69] to extract the most out of these resource-constrained and error-prone quantum computing systems. One of the approaches to do so is by developing hardware- and algorithm-aware quantum circuit mapping techniques to maximize performance. In general terms, mapping refers to the process of modifying potentially hardware-agnostic quantum circuits in such a way that they can be run on a given quantum computing device by respecting all of its constraints while optimizing performance (e.g., algorithm success rate). So far, several mapping techniques have been developed mostly for superconducting and ion-trap qubit devices, as they are nowadays one of the most well-recognized and most-developed qubit implementation technologies in terms of qubit counts and availability to users. However, spin-qubits emerge as a promising technology for scaling up quantum computing systems mainly due to their high integration potential [34, 59, 62, 63, 68, 70]. Therefore, the scientific community is envisioning two-dimensional spin-qubit architectural proposals that could alleviate some of the major challenges toward scalability. Recently, a crossbar array [6] has been experimentally demonstrated showing great promise for architectures with shared control. Such scalable architectural designs come with a new set of hardware constraints for which novel quantum circuit mapping techniques need to be developed.

In this article, we present *SpinQ*, the first native compilation framework focusing on scalable spin-qubit architectures. To this purpose, we target the so-called crossbar architecture proposed in Reference [32]. By creating a deep understanding of its operational constraints, we draw a clear picture of unique mapping challenges that arise in comparison to other qubit technologies. We have devised a novel compilation approach called the *Integrated Strategy*, a method inspired by mapping solutions found in Reference [19, 38]. Rather than seeking pure optimality, this strategy prioritizes scalability to harness the potential of scalable spin-qubit architectures. This pioneering compilation strategy uniquely and effectively navigates the rigid constraints of the crossbar architecture, doing so without adding to the computational complexity in comparison to alternative proposals. Yet, it is important to note that the current iteration does restrict the parallelization of certain gates, indicating room for improvement. However, this design has been created with future advancements in mind, while keeping its time complexity in check. Our aim, through the elucidation of our results, is to highlight the imperative nature of comprehensive performance

evaluation of emerging architectures and mapping techniques. Through our results, we aim to provide key insights into the importance of performing an extensive performance evaluation process of novel architectures and mapping techniques. With this compilation framework, we not only enable quantum algorithm executions on scalable spin-qubit hardware but, perhaps more importantly, we form insights on the behavior and performance of this new breed of architectures. It also offers design guidelines vital for steering future breakthroughs in both hardware and software.

The main contributions of this article are as follows:

(1) an in-depth analysis of mapping challenges to create novel mapping techniques for spin-qubit crossbar architectures;
(2) *SpinQ*, the first native compilation framework dedicated to scalable spin-qubit architectures that utilizes a more scalable compilation strategy compared to previous proposals;
(3) a thorough performance analysis of the main sources of gate/depth overhead and estimated success probability when mapping well-defined quantum algorithms on the crossbar architecture; and
(4) deriving algorithmic- and hardware-specific mapping insights for the crossbar architecture and potentially other spin-qubit architectures from a scalability point of view.

The remainder of this article is structured as follows: In Section 2, the current progress and challenges of scalable spin-qubit architectures are presented. In Section 3, the crossbar architecture is introduced as a potential candidate in scaling quantum devices in two dimensions, as well as its native operations. In Section 4, we comprehensively analyze the unique challenges of mapping quantum algorithms on the crossbar architecture that require novel mapping techniques. Then, in Section 5 we introduce *SpinQ*—the first native compilation framework for scalable spin-qubit architectures. In Section 6, we refer to our experimental methodology, and in Section 7 we thoroughly analyze the performance of *SpinQ* when mapping a broad and well-defined range of quantum algorithms on the crossbar architecture after which we form architectural and mapping insights. In Section 8, we discuss potential improvements of our compilation strategy and we compare its computational complexity to previous proposals. Finally, we conclude our work in Section 9.

2 SPIN QUBITS AS A SCALABLE PLATFORM

To fulfill the promise [48] of quantum computers being machines that solve some classically intractable problems, substantial system sizes have to be reached, i.e., a large number of qubits [1, 16]. It still remains to be seen that qubit implementation technologies (e.g., superconducting, trapped ions, quantum dots, photonics, and defect based on nitrogen-vacancy diamond centers) will succeed in scaling up quantum computing systems with high-quality qubits [13, 50]. Among them, spin qubits in quantum dots are a promising technology for scalable quantum computers due to the maturity of the semiconductor industry, the capability of high integration on a single die compared to other qubit technologies (the physical space of 1 transmon qubit can fit ∼1,000 spin qubits along with classical control electronics), long coherence times (close to 20 μs), and the ability to operate in super-kelvin temperatures (up to 4 kelvin) [12, 13, 20, 34, 59, 62, 62, 63, 67, 68, 70].

Despite the advantages just mentioned, there are still several challenges today toward scaling spin-qubit devices in a sustainable manner. One major challenge is the wiring scheme between the quantum processor and the classical interface, the so-called interconnect bottleneck [59]. Formally, the interconnect bottleneck is described by Rent's exponent [14], which is a measure of optimization in the wiring scheme in both classical and quantum processors. The existing scheme in most quantum devices of having at least one control line per qubit is not scalable in the long term. This is mostly due to the fact that dilution refrigerators have an upper limit to I/O cable capacity and that more cables will progressively make it harder to reach the desired milli-kelvin temperature

due to higher heat dissipation. Therefore, qubit architectures and classical-control electronics have to support multi-qubit shared-control that requires a sub-linear number of control lines alongside an increasing number of qubits. In other words, each control line needs to address multiple qubits to effectively mitigate the interconnect bottleneck when scaling up quantum hardware.

Going a step further, the inability to achieve a scalable wiring scheme also originates from the low device uniformity achieved by today's fabrication tools. In most cases, this implies that qubits cannot be made homogeneous enough to control them effectively in a scalable architecture. The low uniformity results in resonance frequency deviations or other control variations. This means that in an inhomogeneous device a driving signal for a particular operation will have to vary from one qubit to another to get the same outcome [32, 37, 59]. This makes it difficult to successfully control many qubits with the same line, thus contributing to the wiring scheme challenge (i.e., the interconnect bottleneck).

There have been significant efforts [8, 14, 23, 32, 44, 46, 59, 60] to reduce the number of control lines reaching the qubits as devices become ever denser. Such efforts take advantage of the miniaturization capabilities of spin qubits and the large-scale integration of solid-state circuits to address the aforementioned challenges. However, current experimental work has primarily been focused on one-dimensional spin-qubit arrays of small sizes [59], which are not easily scalable. Recently, a 2×2 spin-qubit processor [21] and a 4×4 spin-qubit device based on a crossbar architecture [6, 32] with shared control have demonstrated the potential to scale spin-qubit devices in two dimensions. Therefore, there will be a need, as the technology is advancing and further reducing Rent's exponent, to effectively map quantum algorithms on two-dimensional devices such as the crossbar architecture that comes not only with limited qubit connectivity but also with a new set of constraints. This creates an opportunity to explore its mapping challenges and propose novel solutions.

However, mapping techniques are not studied as much as other qubit technologies such as superconducting and ion traps. In addition, the sample space of experimental devices or architectural proposals is sparse and lacks a detailed description of hardware constraints [7, 23, 60]. Consequently, there is a significant void in evaluation tools specifically designed for benchmarking a range of quantum algorithms. Therefore, it also remains unclear whether existing techniques could be applicable. Then, even if such techniques are realized, then they could be incompatible with existing quantum compilation tools made for other qubit technologies. This could be due to completely different development requirements imposed by the particular spin-qubit constraints and their scalability prospects. In other words, a dedicated compilation framework for spin-qubit architectures with a focus on scalability is still missing. All these obstacles make it difficult to fully explore the possibilities and compare various architectural proposals under relevant application categories.

3 THE CROSSBAR ARCHITECTURE

The crossbar architecture for arranging spin qubits was introduced in Reference [32] as a scalable solution to the interconnect bottleneck. Inspired by the crossbar architecture used in today's classical processors [8, 32], it adopts a similar characteristic, namely shared control. This achieves a quadratic reduction in control lines per qubit [19] and opens up the possibility for high integration of up to 1,000 qubits in a single package.

In this crossbar architecture, qubits are defined by electron spin states in Si-based quantum dots. A Si-based quantum dot is a layer-structured semiconductor device that can confine a single electron with proper gate electrode control after which its quantum-mechanical spin can define a physical qubit [18]. In Figure 1, we illustrate a schematic overview of the crossbar architecture in which every site (circles) represents a quantum dot, some of which are occupied by spin-qubits

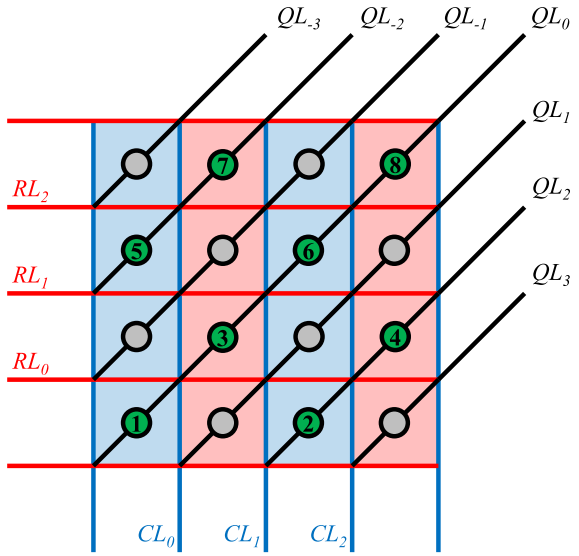


Fig. 1. Schematic overview of the crossbar architecture and operational control lines [32].

(numbered, green circles). Spin qubits are usually sparsely initialized in a pattern to reduce potential cross-talk and to allow for long-range entanglement through shuttling qubits across the array [32]. In this case, a checkerboard pattern provides these benefits. Finally, the crossbar architecture requires high fabrication uniformity of its materials to minimize operational errors. It is possible, however, to mitigate such errors or even vanish them by operating the crossbar at low magnetic fields and with proper tuning (e.g., separated resonance frequencies between columns). Furthermore, a crossbar module is envisioned to be self-contained and duplicated in a network of multiple crossbar modules. This can provide the means to realize quantum error correction in large-scale systems enabled by fast-shuttling, low-error communication links.

It is now important to focus on the three different kinds of shared control lines used to perform operations on qubits: vertical (column line, CL), horizontal (row line, RL), and diagonal (qubit line, QL). Notably, each line affects all the sites that it is connected to. For instance, in Figure 1 line QL_{-2} affects the sites in which spin-qubits 5 and 7 reside. This imposes some particular restrictions in the parallelization of instructions, which we will discuss in Section 4. Below, we will abstractly describe the control properties for executing native gates of the crossbar architecture. Note that any non-native gates can be decomposed into native ones as explained in Section 5.1.1. A more detailed explanation is provided in Reference [19].

3.1 Qubit Shuttling

In the crossbar architecture, qubits can be moved around by performing shuttling operations. In a shuttle operation, vertical or horizontal lines are used as barrier gates, depending on the direction. Lowering or raising these barriers can create pathways from which qubits can move (shuttle) orthogonally from one site to another with the use of **Direct Current (DC)** signals through the diagonal lines. Specifically, when a barrier separating a qubit and an empty quantum dot is lowered then it can move pushed/attracted by the voltage difference of the QL lines going through the origin and destination sites. Figure 2(a) shows an example of shuttling, in which qubit 3 is moving one site to the left. The order in which the control line signals should be pulsed and other requirements are analyzed in Section 4.1. Figure 2(b) shows the potential parallel shuttle operations to shuttling

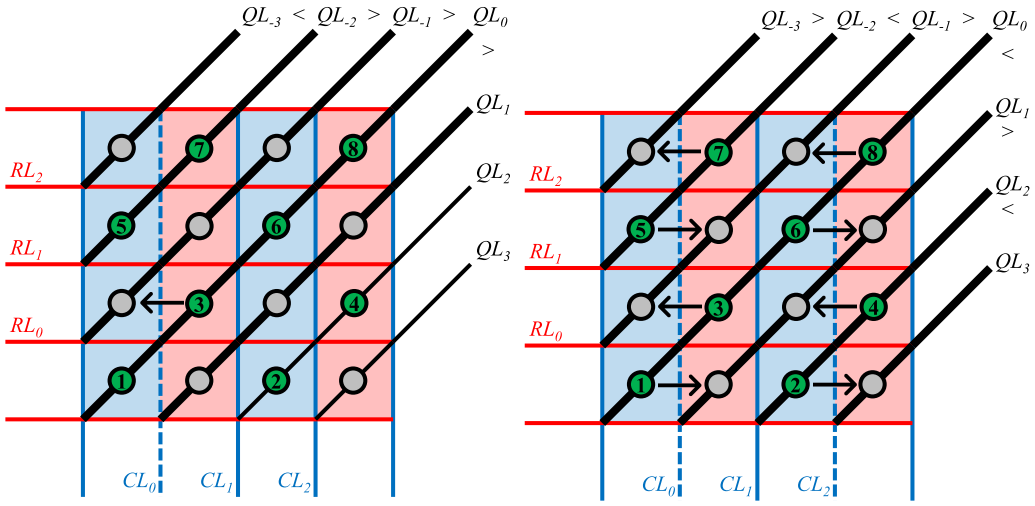


Fig. 2. (a) Shuttling example of qubit 3 moving one site to the left. The barrier CL_0 between origin and destination site is lowered and voltage of QL_{-1} is larger than QL_0. The order and signal requirements will be further explained in Section 4.1. (b) The potential of parallelizing shuttling operations that can be executed in parallel with shuttling left qubit 3. However, not all qubits can be shuttled arbitrarily at the same time due to the specific requirements or potential conflicts caused by mapping in this architecture. These challenges are analyzed in Section 4.

qubit 3 left. To do that, the requirements of shuttling qubit 3 need to be compatible with the others and satisfied in the same order. Note that the larger the crossbar topology the more difficult it gets to parallelize shuttle operations. A deeper analysis of parallelization possibilities and challenges is given in Section 4. Although this architecture can support gate-based communication with two subsequent $\sqrt{SWAP}$s (see Section 3.2), resulting in a $SWAP$ gate similar to superconducting qubits, shuttling qubits is preferred due to higher operation fidelity and shorter execution time. It should be noted that shuttling horizontally, i.e., between columns, causes a Z rotation (see Section 3.3) that should be mitigated by timing well the next operation(s) [32].

3.2 Two-qubit Gates

Two two-qubit gates are supported by the crossbar architecture, $CPHASE$ and $\sqrt{SWAP}$, with the latter being chosen for this work due to its higher operational fidelity and faster execution time according to Reference [32]. A $\sqrt{SWAP}$ can be performed similarly to the requirements of a shuttle operation, analyzed in Section 3.1 and 4.1. The only requirement differences are that the operant qubits need to be vertically adjacent (i.e., same column) and the fourth requirement in Section 4.1, related to the QL lines going through the two sites, need to have equal voltages. Once these are satisfied, similar rules to shuttle parallelization are applied for parallel $\sqrt{SWAP}$s as explained in Section 3.1. Finally, it is possible to parallelize two-qubit gates and shuttle operations as long as all their constraints are satisfied.

3.3 Z Rotations

In the crossbar architecture, single-qubit gate rotations should be separated into two categories: Z rotations and X or Y rotations.

Z qubit rotations can be controlled by a well-timed qubit shuttling to and from a neighboring column [19, 32, 38]. Due to the differences in Zeeman energies between the two column parities,

timing is key. The imposing alternating magnetic fields on qubits rotate them in the Z axis, and the longer they stay in the opposite column parity the more they rotate. Therefore, when the second shuttle is timed purposefully, the qubit can return to the initial position rotated at any angle. Besides this timing peculiarity between the two shuttles, parallelization constraints and mapping challenges are the same as qubit shuttling. Finally, it is possible to parallelize Z rotations, two-qubit gates, and shuttle operations in the same cycle when all requirements are satisfied.

3.4 X or Y Rotations

As for X or Y rotations, either all qubits belonging to red-colored columns or all qubits in blue-colored columns are rotated (see Figure 1). This is called semi-global single-qubit rotation and is implemented by electron-spin-resonance [32, 61]. A high-level representation of how a particular column parity is addressed is given in Figure 4(a). Depending on the duration of the applied magnetic field at the CL lines, the qubits can be rotated at any angle.

3.5 Measurement

The readout process allows for local single-qubit measurements by using the Pauli Spin Blockade process [15]. With this process, the measurement outcome is determined by whether a qubit shuttle toward a horizontally adjacent ancilla qubit was successful. In this work, we considered an ideal measurement process in which no ancilla qubits are involved.

4 QUANTUM CIRCUIT MAPPING CHALLENGES OF THE CROSSBAR ARCHITECTURE

The mapping process of a quantum circuit plays an essential role in the successful execution of algorithms on a quantum computer. It consists of a cascade of routines that transform a (potentially hardware-agnostic) quantum circuit to a hardware-compatible version. However, current NISQ quantum processors are severely constrained and cannot run useful applications successfully yet, despite notable efforts in this field.

Examples of hardware constraints are low qubit connectivity, cross-talk, reduced primitive gate set, low coherence time, fabrication imperfections, and limited classical-control resources. Therefore, a mapping process needs to consider such limitations and try to optimize performance as much as possible to increase the algorithm's success rate. So far, there are a plethora of proposed solutions that differ in strategy, methodology, and performance metrics to optimize [4, 28, 29, 31, 39, 41, 42, 47, 49, 54, 55, 69].

Such mapping techniques have been mostly developed for superconducting and ion-trap qubit devices. However, as of now, there is not much focus on spin-qubit architectures and their particular characteristics. Although spin-qubits are now in a rather early development stage, their scalability potential is undeniable, and therefore it is timely to lay grounds for developing novel mapping techniques and inspire further research. As previously mentioned, in this work we focus on the crossbar architecture that comes with a unique set of constraints that affect the parallelization of quantum operations, the application of X or Y rotations on individual qubits, and the routing of qubits (i.e., moving qubits around the topology).

4.1 Parallelization of Quantum Operations

Most of the operation parallelization restrictions of the crossbar architecture come from the fact that control lines are shared among multiple qubits and each line has a specific role and relation to one another. It should also be mentioned that most operations must be implemented with strict pulse durations and time intervals, depending on the addressed site [19, 32]. Although such pulse durations have to be carefully considered in the mapping process by providing recent calibration

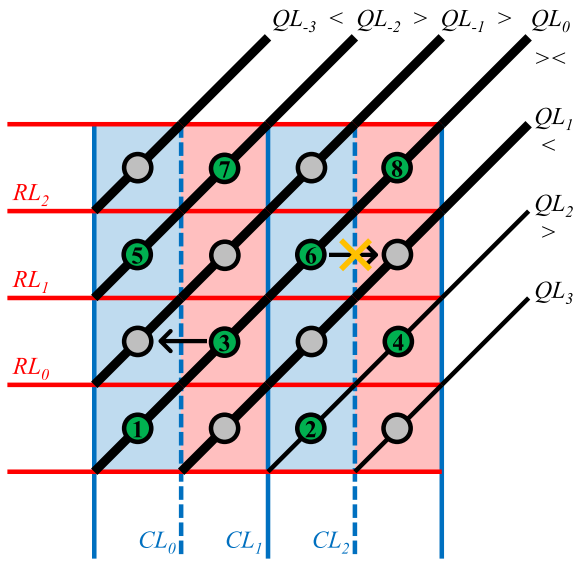


Fig. 3. Parallelizing shuttles of qubit 3 and 6 is not allowed due to violation of constraints shown as $QL_0 >< QL_1$.

data [42, 54, 55], in this work we consider an ideal crossbar architecture, as such data are not available yet. Despite that, the mapping techniques proposed in this work are compatible with similar considerations and can be added once calibration data are available.

To better illustrate what conditions and constraints there are when trying to parallelize quantum operations, let us consider the following example in which two shuttles are tried in parallel. As shown in Figure 2(a), the following requirements must be fulfilled in that order to shuttle qubit 3 one site to the left:

(1) The destination site must not be occupied by another qubit.
(2) The barrier between destination and origin sites must be lowered. This is depicted as a dashed vertical CL_0 line.
(3) All barriers surrounding the origin and destination sites must be raised. This is shown as solid red RL (RL_0 and RL_1) and blue CL lines (CL_1 and the always-raised most-left CL line).
(4) The voltage going through the QL line of the destination site (QL_{-1}) must be higher than the one going through the origin site (QL_0). This is shown as $QL_{-1} > QL_0$ in the top-right of Figure 2(a).
(5) To prevent other qubits in these two columns from shuttling, the voltage going through their QL lines must be higher than their adjacent empty sites. This is depicted as voltage level relations between QL lines. Note that QLs with no voltage relations are irrelevant for this particular shuttle operation.

Now, we assume a shuttle of qubit 6 to the right (as depicted in Figure 3) in parallel to the left shuttle of qubit 3. This implies that all previously listed requirements of qubit 3 need to be satisfied along with the new ones of qubit 6. However, the fourth requirement cannot be satisfied, as the $QL_0 > QL_1$ relation we had before would have to be changed to $QL_0 < QL_1$. If this change is allowed, then we violate the fifth requirement of the first shuttle and, as a consequence, qubit 1 will shuttle to the right. Therefore, we cannot shuttle qubits 3 and 6 as such at the same time. Contrary to that, in Figure 2(a) we were able to shuttle qubit 6 only, because qubit 1 shuttles to the right at the same time.

Thus, we see that scheduling parallel gates in the crossbar implies a strict simultaneous satisfaction of all signal requirements for each gate. It also depends on the specific gate (operation) set to be parallelized and their corresponding qubit positions on the topology. Any violation of the above conditions would potentially result in the shuttling of unwanted qubits, unwanted qubit interactions, or unknown qubit states. As seen in the previous example, performing quantum operations in parallel without affecting other qubits and meeting all signal requirements is not always possible regardless of qubit distance. In fact, it does not matter how far qubits are away from each other but whether control lines are shared between them or not and whether their operational requirements and relations match. A collision is another conflict that could be caused by improper parallelization of shuttling gates that try to move two qubits toward the same site. Unlike more popular qubit architectures based on superconducting or ion traps, this form of operational constraint is unique. On the one hand, sharing control lines tackles the interconnect bottleneck; on the other hand, it intrinsically constraints its parallelization capabilities.

Finally, in other qubit architectures, it is possible to perform different gate types in parallel. In the crossbar architecture, this is not always the case. For example, applying single-qubit gates and shuttling operations at the same time is not possible, because, in the former, CL lines need to carry an **alternating current (AC)** signal (see Figure 4(a)), while the latter require DC signals for raising or lowering the barriers.

4.2 Mapping of X or Y Rotations on Single Qubits

As established in Section 3, X or Y qubit rotations are implemented semi-globally, meaning that either all qubits in odd or even column parities will be rotated. However, during an arbitrary cycle, not all qubits in odd or even columns should be rotated. Note that the notion of a cycle refers to the basic unit of time representing one step in a sequence of quantum gates applied to a set of qubits, and it may contain multiple gates. Therefore, to compensate for unwanted X or Y rotations, one has to come up with a specific rotation mapping scheme such that only the targeted qubits are rotated. In this work, we have implemented the scheme introduced in Reference [19]. We illustrate how it works in Figure 4, in which we are interested in rotating only qubit 5. This is another unique characteristic of this architecture, as additional routing is needed to perform single-qubit rotations on specific qubits thus imposing new challenges to the mapping process.

4.3 Routing of Qubits

We will now expand specifically on the qubit routing challenges.

Routing a qubit in the crossbar means that an electron (or a hole, depending on the materials) is physically "pushed" to an empty site (i.e., an empty quantum dot). This mechanism is similar to a **quantum charged-coupled ion trap device (QCCD)** where ions are shuttled through a common channel from trap to trap, assuming sufficient destination ion trap capacity [10, 40]. The QCCD architecture and the crossbar architecture fundamentally differ in topology, but both require special algorithms or additional routing routines to maintain control of qubit positions and avoid potential conflicts. The topology of the crossbar is essentially a two-dimensional (2D) square grid, whereas a QCCD device resembles a bi-linear array with an "H" shape and in its corners ion traps are located—each dedicated to a specific purpose during operation. The particular shape of a QCCD device creates different constraints for moving qubits or parallelizing operations compared to the crossbar architecture hence their mapping techniques are different even though both use shuttling.

Shuttling qubits on the crossbar not only depends on specific control signal requirements and available empty sites but also on other qubit positions. We illustrate this fact with an example in Figure 5, in which a vertical shuttle operation of qubit 3 is indicated by a black arrow. In this case, the horizontal barrier RL_0 has to be lowered and the QL lines have to be pulsed in certain

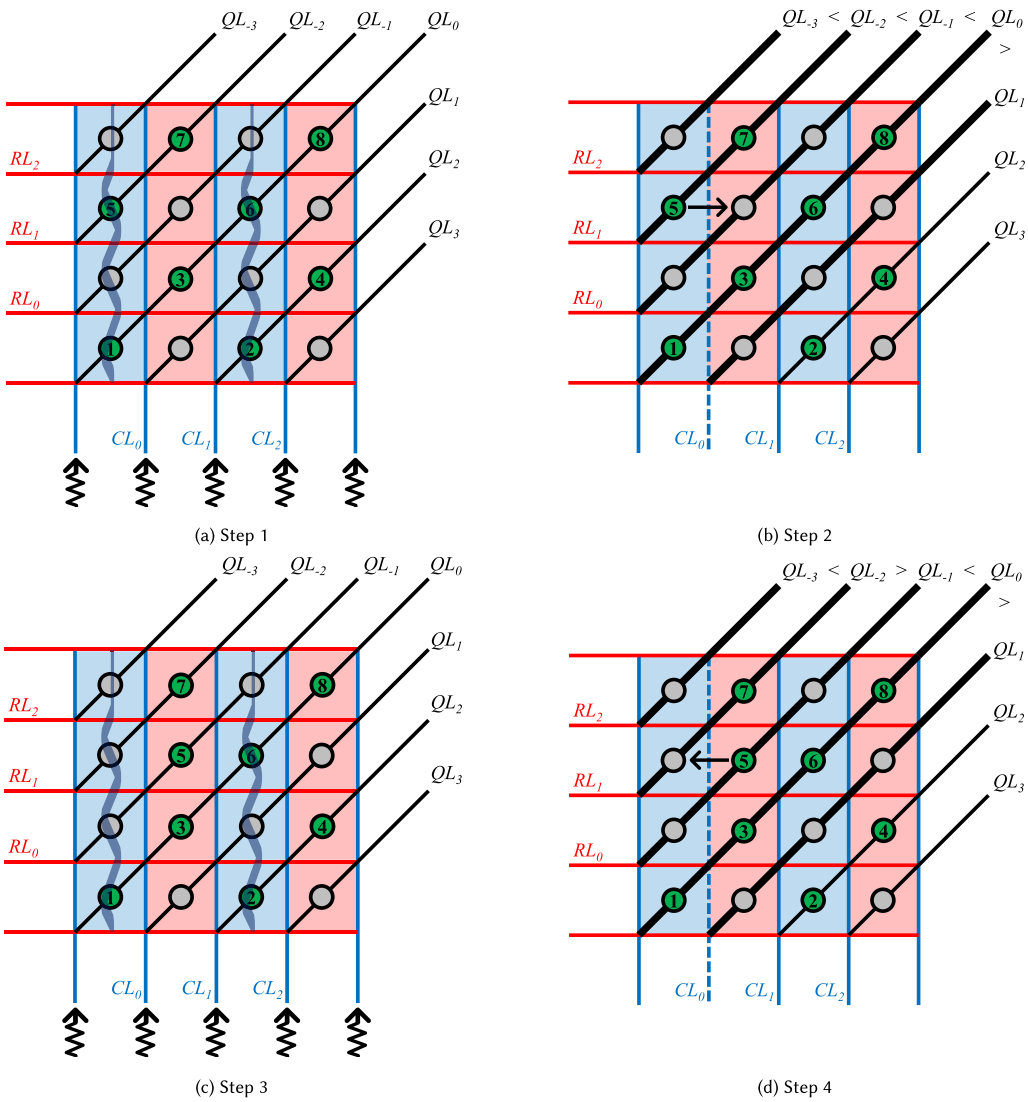


Fig. 4. Single-qubit gate on qubit 5. (a) Step 1: AC signals through the CL lines induce magnetic fields on qubits 1, 5, 6, and 2 belonging to the even column parity, thus rotating their state. The direction and frequency of these signals determine which columns (red or blue) and what rotation (X or Y) will be applied to the corresponding qubits [32, 61]. (b) Step 2: The targeted qubit 5 is moved with a shuttle operation to a different column parity. For this operation, CL_0 opens and closes as a barrier and the relevant diagonal lines (QL) create potential gradients to only allow for qubit 5 to move (shuttle). Note that to shuttle only qubit 5, all relevant QL lines need to have voltage relations with one another. (c) Step 3: An inverse rotation is applied again in all even columns containing qubits 1, 6, and 2, similarly to Step 1. (d) Step 4: Target qubit 5 is moved with a shuttle operation to the initial position.

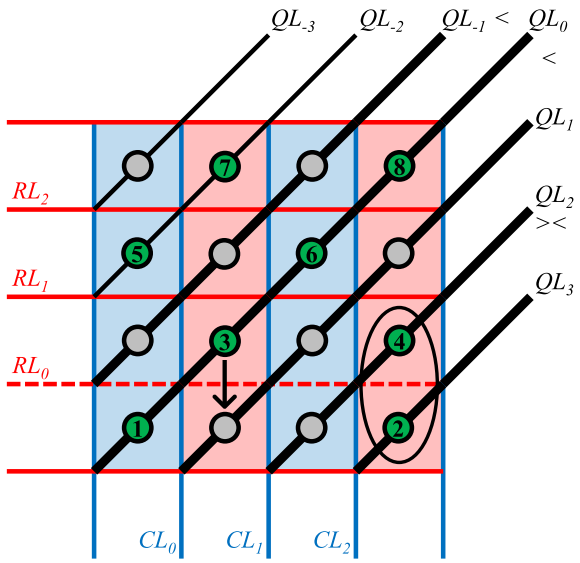


Fig. 5. Example of a conflict: The operational requirements of shuttling qubit 3 downwards have lowered the RL_0 barrier, thus causing an unwanted interaction between qubits 2 and 4. Additionally, QL_2 and QL_3 lines passing through these qubits need to satisfy the fifth requirement described in Section 4.1 and by doing so create a violation $QL_2 >< QL_3$. Note that QL_1 and QL_2 are signaled in this example but do not need to have a voltage relation requirement between them.

voltage relations to allow for correct shuttling of only qubit 3. However, an unwanted interaction is caused between two other qubits in the same rows (qubits 2 and 4, circled), regardless of the QL_2 and QL_3 relation. Analogously, the same issue exists with a horizontal shuttle when having two horizontally adjacent qubits in the same columns where the shuttle takes place [19, 38]. Last, there can be a blocked path conflict where there is no empty site for a qubit to shuttle to.

Therefore, a dedicated qubit routing algorithm for the crossbar architecture has to be developed to avoid collisions, blocked paths, and unwanted interactions. Furthermore, even if we had such a dedicated routing algorithm, then the same conflicts have to be considered when rearranging gates in parallel during scheduling. For that, control signals and qubit positions must be carefully monitored within the mapping process. We provide a summary of unique architectural features and operational constraints in Table 1 to clearly show these unique constraints. From the description above, it is clear that both the routing and scheduling processes need to jointly work in a strategy to avoid conflicts and optimize for algorithm success rate. This will be the main characteristic of *SpinQ*, presented in the following section.

5 *SpinQ*—THE FIRST NATIVE COMPILATION FRAMEWORK FOR SCALABLE SPIN-QUBIT ARCHITECTURES

In this work, we present the first native compilation framework (*SpinQ*) dedicated to compiling and mapping quantum circuits onto scalable spin-qubit architectures, such as the previously described crossbar. We have based our mapping techniques on previous works from References [19, 38] while improving them from a scalability standpoint.

Figure 6 shows the schematic structure of our framework. As *input*, *SpinQ* accepts **Quantum Assembly Language (QASM)** format files that describe quantum circuits (used as benchmarks) in a device-independent manner. To increase the flexibility of our framework, particular characteristics of the crossbar architecture can be defined in an *architectural configuration* file. It can

Table 1. Summary of Unique Architectural Features and Operational Constraints of the Crossbar Architecture

Features of the architecture \ Constraints	Parallelization Constraints	Qubit Routing Constraints
Shared control lines	✓ Need for specialized parallelization algorithm to prevent conflicts (collisions, blocked paths, unwanted interactions) ✓ Limited due to strict simultaneous control signal requirements satisfaction, cycle gate set and qubit positioning (see Section 4)	✓ Need for specialized algorithm to prevent conflicts based on qubit positions while respecting all operational requirements
Semi-global single-qubit gates	✓ Possible routing conflicts from specialized mapping scheme (see Section 4.2) ✓ Only one rotation axis and angle at one column parity allowed in the same cycle	✓ Need for specialized mapping scheme with conflict-free routing (see Section 4.2)
Shuttling for movement	✓ Limited due to shared control constraints ✓ Need for specialized parallelization algorithm to avoid conflicts ✓ Only two-qubit gates qubit gates and Z rotations allowed in the same cycle	✓ Strict control and timing constraints (see Section 3.1 and 4). ✓ Need for specialized algorithm to prevent conflicts during routing
Z rotations with shuttling	✓ Limited due to shared control constraints ✓ Need for specialized parallelization algorithm to avoid conflicts ✓ Only shuttling and two-qubit gates allowed in the same cycle ✓ The second time-sensitive shuttle needs to be scheduled immediately after the first (see Section 3.3)	✓ Need for specialized algorithm to prevent conflicts during routing for both shuttle operations (see Section 3.3)

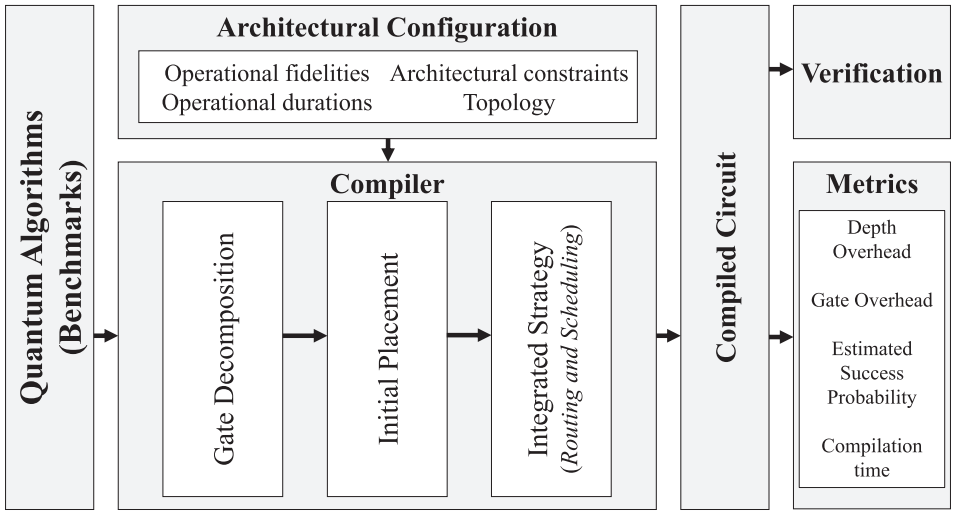


Fig. 6. Overview of our SpinQ framework proposed in this article.

include custom operations and their particular attributes such as gate durations, mathematical description of the unitary matrices, associated gate fidelities, and architectural constraints, among others. Moving on, the *compiler* consists of a series of passes to decompose gates, route qubits, and schedule instructions. To address the unique mapping constraints of the crossbar architecture, we have conceptualized and developed the *Integrated Strategy*. The current implementation has room for improvement (see Section 8); however, our aim in this work is to study the behavior

of algorithms to form deep insights about novel mapping techniques and spin-qubit architectures from a scalability perspective. The compiler's output is a QASM file of the *compiled circuit* that is compatible to be executed on the given architecture. Optionally, a *verification* step can take place to ensure the compiled circuit meets all operational constraints of the architecture without any conflicts. This step is implemented to be able to check the compatibility of architectural proposals that are not physically realized yet, such as the crossbar [32]. Finally, several *performance metrics* are extracted from the compiled circuit to evaluate algorithm performance. In the next sections, we will further discuss each of the compiler elements.

5.1 Compilation Passes

The compiler consists of the following steps:

5.1.1 Decomposition of Quantum Gates. Inputted QASM quantum circuits are first transformed into a custom-made intermediate representation data format. Quantum gates are then decomposed into gates native to the architecture based on the decomposition sequences specified in the architectural configuration file. These sequences are reported in Reference [38].

5.1.2 Physical Initialization of Spin Qubits. A checkerboard pattern has been proposed [31] to allow space for data and ancilla qubits to move [19, 38]. The physical space achieved between the qubits not only facilitates shuttling that avoids possible conflicts but also reduces crosstalk and enables surface code error correction [32]. As we will discuss later, maintaining this placement pattern throughout a circuit execution plays an integral role in the *Integrated Strategy*. Having said that, initializing qubits in alternative patterns and changing them during execution is possible. This flexibility offered by the spin-qubit technology can be particularly advantageous to highly specialized mapping techniques for the crossbar as well as for other architectural proposals.

5.1.3 Virtual-to-physical Qubit Initial Placement. The current version of *SpinQ* associates virtual qubits of an algorithm with physical qubits in a one-to-one manner by numbering the physical qubits from left to right and from bottom to top as shown in Figure 1. In the results sections 7 and 8, we will provide insights on how common initial placement algorithms can be adapted to improve the performance of spin-qubit architectures, such as the crossbar.

5.1.4 Integrated Strategy for Routing and Scheduling. As explained in Section 4, both routing and scheduling techniques must avoid conflicts. To do that, a specific strategy needs to be conceptualized. There can be various strategies with various performance and compilation time tradeoffs. The presented *Integrated Strategy* tilts toward minimizing compilation time while having great prospects to be competitive against other strategies that focus on algorithm performance, as will be discussed in Section 8.

To begin with, in the *Integrated Strategy*, the checkerboard pattern qubit placement [32], also known as "idle-configuration" in Reference [19], should be maintained as much as possible. This provides at least two empty sites for every qubit to move toward to.

When routing for two-qubit gates, we maintain the checkerboard pattern throughout the circuit execution with a conflict-free shuttle-based SWAP technique [38] as shown in Figure 7. Note that this movement of qubits results in a gate overhead of 4 (i.e., four shuttle operations), but a depth overhead of 2, as these two shuttle pairs can always be executed in parallel. To bring one of the qubit operants to the appropriate position before the two-qubit interaction, multiple shuttle-based SWAPs might be performed. For that, we have implemented a shortest-path algorithm based on the Manhattan distance between the qubit operants. Once the two qubits are in the shortest position possible, the next step is a horizontal shuttle of one of them, either to the left or to the right, after which the target and control qubits are vertically adjacent, and the checkerboard pattern

ALGORITHM 1: Integrated Strategy

Input: Intermediate representation *ir_d* of decomposed circuit
Output: Intermediate representation *ir_c* of compiled circuit

Initialize *ir_r*, *ir_c_1*, *ir_c*
for *gate* in *ir_d* **do** ▷ The shuttle-based SWAPs. See Figure 7
  if *type(gate) = two-qubit* gate and *qubits(gate)* not neighbors **then**
    ir_r ← new cycles with shuttle operations based on the shuttle-based SWAP process ▷ See Section 5.1.4
    ir_r ← new cycle for shuttle that makes *qubits(gate)* vertically adjacent
    ir_r ← new cycle for *gate*
    ir_r ← new cycle for shuttle that restores the checkerboard pattern
  else
    ir_r ← new cycle for *gate*
  end if
end for
for *gate* in *ir_r* **do** ▷ First pass
  if *type(gate) = z_rotation* **then**
    ir_c_1 ← new cycle with *gate*
  else if *type(gate) = shuttle* of shuttle-based SWAP **then** ▷ See Figure 7 and Section 5.1.4
    if *one of the shuttle pair* in *ir_c_1[cycle_index]* **then**
      ir_c_1[*cycle_index*] ← add *gate*
    else
      ir_c_1 ← new cycle with *gate*
    end if
  else if *type(gate) = two-qubit gate* **then**
    ir_c_1 ← new cycle with *gate*
  else if *type(gate) = X or Y rotation* **then**
    condition ← is there a *cycle_index* in *ir_r* with the same rotation on qubits belonging in the same column parity as *gate* and satisfy gate dependencies?
    if *condition = True* **then**
      ir_c_1[*cycle_index*] ← add *gate*
    else
      ir_c_1 ← new cycle with *gate*
    end if
  end if
end for
for *gates* in *ir_c_1* **do** ▷ Second pass
  if *type(gates) = shuttle* **then**
    ir_c ← new cycle with *gates*
  else if *type(gates) = two-qubit gates* **then**
    ir_c ← new cycle with *gates*
  else if *type(gates) = z_rotation* **then**
    shuttle_direction, *shuttle_opposite_direction* ← left or right (depending on constraints)
    ir_c ← new cycle with *shuttle_direction*
    ir_c ← new cycle with *shuttle_opposite_direction*
  else if *type(gates) = X or Y rotation* **then**
    non-problematic gate set, *problematic gate set* ← check if mapping scheme of Section 4.2 is applicable
    ir_c ← new cycles for each gate of the mapping scheme for the *non-problematic gate set*
    if *problematic gate set* ≠ *Null* **then**: ir_c ← new cycles for each gate of the mapping scheme
  end if
end for

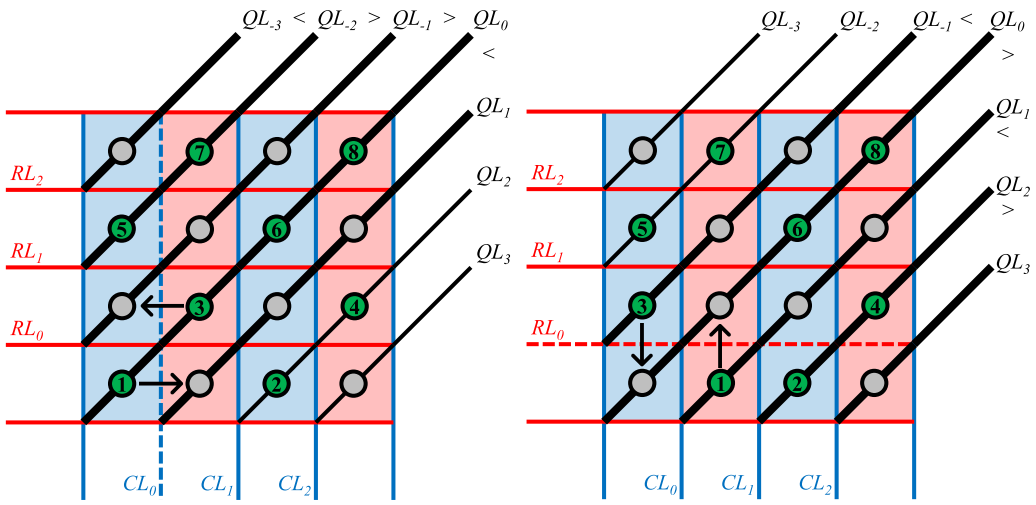


Fig. 7. Conflict-free shuttle-based SWAP for two-qubit gate routing: With this technique, two diagonally neighboring qubits exchange their position by consecutively performing two horizontal and two vertical shuttles. Each pair can be performed in parallel.

is temporarily broken. Proceeding the $\sqrt{SWAP}$, a final shuttle returns the qubit to the previous position, and the checkerboard pattern gets restored. Note that the aforementioned process can be successfully executed only in that particular order; otherwise, there can be a routing conflict. Overall, routing for two-qubit gates requires at least one shuttle-based SWAP and exactly two horizontal shuttles.

So far, we have only talked about routing for bringing together qubits for performing two-qubit gates. However, qubit routing is also needed for shuttle-based Z rotations and might be needed for X or Y rotations, as discussed in Sections 3.3 and 4.2, respectively. Mapping these two categories of gates, therefore, should always respect operational requirements and avoid conflicts. This also means that the "idle configuration" should be maintained when routing for these gates, as well. Thus, the second consideration of the *Integrated Strategy* is the integration of single-qubit gate routing within the scheduling stage, hence the name "integrated," to prevent conflicts and optimize performance.

The *Integrated Strategy* continues with two passes. In the first pass, the scheduler tries to parallelize X or Y gates in an ideal manner, based on the gate dependencies [29] (ignoring any potential conflicts) and Z gates individually. This is no different than other single-qubit gate scheduling processes proposed for other qubit architectures. However, it differs on the second pass that integrates the routing procedures for X, Y, and Z gates. The second pass iterates over each cycle produced by the first pass. For each cycle, there are two causes: (a) If no conflicts are detected when scheduling the shuttle instructions of the mapping scheme described in Section 4.2, then these instructions are inserted, each in a new cycle one after the other (b) if conflict(s) are detected, the subset of the problematic gate(s) is separated. Once the non-problematic gate subset is scheduled according to case (a), the problematic subset is recalled. This time it constitutes a conflict-free cycle and is scheduled similarly to case (a). The *Integrated Strategy* is described in Algorithm 1, and its time complexity is calculated to be $O(n)$, with n as the number of gates.

The key concept of this strategy is the ideal parallelization in the first pass that is aimed to relieve the increased complexity of concurrently avoiding conflicts and optimizing. Then, the second pass

tries to satisfy the scheduling of the first pass in the least cycles possible while completing the mapping steps of all gates (e.g., applying the mapping scheme for X or Y gates or adding shuttles for Z rotations, etc.). Overall, this first implementation of our *Integrated Strategy* does not parallelize gates of different types in the same cycle, and thus each cycle is dedicated to one instruction type. Additionally, it leaves room for improvement while maintaining its $O(n)$ time complexity, as discussed in Section 8.1. Fortunately, the strategy described above and suggested extensions in Section 8.1 can be adapted to a real setup. As explained in Section 4, a fabricated crossbar device will most likely have material imperfections, thus requiring pulse calibration per site. As pointed out in References [19, 26], pulsing control lines prematurely to account for material variations could cause an unwanted interaction. Since, however, the *Integrated Strategy* (or an extension thereof) exclusively schedules gates of the same type in each cycle, fine-tuning pulses within is possible before moving to the next cycle.

5.2 Performance Metrics

We will now introduce the metrics used in this work to evaluate the performance of *SpinQ* when mapping different algorithms on the crossbar architecture.

5.2.1 Gate Overhead. One commonly used metric to evaluate the performance of a mapper and its underlying architecture is gate overhead. We calculate it as the percentage relation of additional gates inserted by the mapper to the number of gates after decomposition. We do not count decomposition gate overhead, as it is always proportional to the number of gates. Getting a clear view of the various sources of gate overhead will help to form useful insights. Therefore, the main sources of gate overhead are the following:

- four additional shuttle instructions per shuttle-based SWAP for two-qubit gates
- At least three additional instructions for each X or Y rotation gate due to the semi-global rotation scheme (Section 4.2)
- two additional shuttle instructions for each two-qubit gate
- one additional shuttle operation for each Z rotation gate.

Note that, unlike superconducting architectures where gate overhead results from routing instructions (i.e., SWAP gates) for performing two-qubit gates, in the crossbar, it can be caused by single-qubit gates as well.

5.2.2 Depth Overhead. Another commonly used metric to evaluate the performance of a mapper and its underlying architecture is the depth overhead of a circuit. The depth of a circuit is equal to the minimum number of timesteps of a circuit when executing gates in parallel [5, 22, 29, 52, 69]. Note that the initial circuit depth is calculated after scheduling the circuit only by its gate dependencies, meaning without any architectural constraints. We calculate depth overhead as the percentage relation of additional depth produced by the mapper to the circuit depth after decomposition. The main sources of depth overhead are as follows:

- At least three additional cycles for each X or Y rotation gate due to the semi-global rotation scheme (Section 4.2)
- Two additional cycles per shuttle-based SWAP for two-qubit gates
- Two additional cycles for each two-qubit gate
- One additional cycle for each Z rotation gate

5.2.3 Estimated Success Probability. A key metric to assess the performance not only of the compiler but also, in general, of a quantum computing system is the algorithm's success rate. From an experimental point of view, the algorithm success rate is calculated by executing the

algorithm several times on a given (real) quantum processor and creating the distribution of successful executions, based on the expected measurement. An alternative way to calculate the success rate without the need for a real quantum processor is by classical simulation. Recently, hybrid Schrodinger-Feynman simulations have been used efficiently but only for shallow circuits [11, 36, 43]. Another method uses tensor networks and has been shown as a more scalable technique for IBM's Eagle or Google's Sycamore chips but suffers from exponential time complexity on more connected architectures or circuits larger in both depth and width [43, 56].

However, there is a need for a more efficient method able to approximate the success rate of much larger circuits. One of the most commonly used methods is considering the final compiled circuit and particular architectural configurations given as input at the beginning of compilation [49, 58]. The **estimated success probability (ESP)** of an algorithm can be calculated as

$$ESP = \prod_i \prod_j gate\_fidelity_{i,j}, \tag{1}$$

where i represents the ith timestep and j the jth gate in the ith timestep.

This method is less complex in both time and space but not as accurate, compared to classical simulations. From Equation (1), it is evident that the time complexity of ESP only increases linearly with the number of gates in a circuit while space complexity remains constant, contrary to the aforementioned methods. To expand it, we have considered a per-type and per-location variability of gate fidelities based on a normal distribution. This implies that, for instance, a two-qubit gate (e.g., $\sqrt{SWAP}$) will have lower fidelity than a single-qubit gate and that the actual fidelity will depend on the exact location in the topology. These expansions constitute a more realistic, i.e., closer to a real device, estimation of circuit success probability,

$$ESP = \prod_i \prod_j gate\_fidelity_{i,j}^{x,y}, \tag{2}$$

where i represents the ith timestep, j the jth gate in the ith timestep, and and x, y are the physical qubit(s) coordinates.

5.2.4 Compilation Time. In this work, we are interested not only in building mapping techniques themselves but also in their scalability potential. This necessitates that our proposed *SpinQ* strategy should remain efficient for a variety of quantum circuit parameters (e.g., number of qubits or percentage of two-qubit gates). By measuring the compilation time for mapping quantum circuits, we get a reference of the scalability of our implementations.

5.3 Verification

A verification tool is important to this work due to the lack of a working device for real-system testing. It is used on demand in the initial stage of development to debug and verify current or future mapping approaches. The tool searches for mismatches between the qubits' position history stored during the compilation and all shuttling sequences. This ensures that all routing instructions added in the final compiled circuit will shuttle the right qubits in the correct places without conflicts and vice versa. This is critical for an architecture such as this one where both the routing and scheduling can produce conflicts. It also checks for operational constraint violations and potential conflicts caused by those. Finally, and after the previous checks, a state vector simulation takes place between the main stages of the compiler with the use of Qiskit Aer library [25]. Specifically, it compares the probability distributions produced by the *qasm_simulator* backend between the initial circuit, the decomposed, the routed for two-qubits gates, and the one processed by the *Integrated Strategy*. This ensures that the mapping techniques and the compilation strategies used do not change the outcome of the algorithm. However, it should be noted that in

non-application-based algorithms (e.g., randomly generated) their state distribution probability can be anything and will suffer a change just from the decomposition stage compared to application-based algorithms. For this reason, this last verification stage cannot be used for all benchmarks. Additionally, this verification cannot be used for more than 30 qubits due to exceeded memory requirements.

6 EXPERIMENTAL METHODOLOGY

6.1 Benchmarks

We have generated 3,630 random uniform algorithms [53] containing X, Y, Z, and $\sqrt{SWAP}$ gates (all native to the crossbar architecture) to be used as benchmarks. With this set, we can vary on demand the number of gates, number of qubits, and percentage of two-qubit gates. For example, a random uniform benchmark with 50% of two-qubit gates relative to single-qubit gates will have 33.33% of X or Y gates, 33.33% of Z gates, and 33.33% of two-qubit gates. Generating synthetic circuits provides a well-controlled benchmark collection from which we can better understand results and form insights. Moreover, we use real benchmarks from the RevLib library in a [5–1400] gate range [64]. Quantum circuits from this library are often used in related quantum circuit compilation works [28, 39, 69], and it consists of quantum algorithms with parameters ranging from 3 to 16 qubits, 18.75% to 100% of two-qubit gates, and 5 to 512,064 gates. Finally, we also consider quantum circuits from the Qlib library [33], which contains real quantum algorithms in increasing sizes.

6.2 Benchmarks Characterization

When it comes to performance evaluation, it is important to not only consider properties of the architecture but also the characteristics of quantum circuits. The simplest and most commonly [4] used parameters of quantum circuits are number of qubits, number of gates, and absolute or relative (i.e., percentage) number of two-qubit gates. However, only these three characteristics can be misleading for two reasons. First, two benchmarks, for instance, could have the same parameter values but heavily differ in the circuit's structure [4]. If one of the algorithms has all qubits interacting with every other qubit, the routing will inherently be higher than the second algorithm which has the same number of interactions, but with only one qubit pair interacting. The structure of a quantum circuit is derived from its **qubit interaction graph (QIG)**, which represents the number and distribution of interactions (i.e., two-qubit gates) between virtual qubits. Several internal circuit parameters can be extracted from the QIG that better distill its properties [4]. Having said that, we analyze QIGs mainly visually, as this is still an active field of research [4]. We support these conclusions by extracting the average *degree* [4] or *program communication* [57] of the QIG, which represents the average number of edges that are incident to (i.e., connected to) a node. In simple terms, it expresses the level of "connectedness" of a graph. We can thus make concrete conclusions and form insights from such a QIG assessment. The second reason is that initial gates can be decomposed to natively supported instructions for the underlying architecture. This means that the number of gates and ratios (percentages) between each gate type can differ from the initial set to the decomposed set, meaning that evaluations can become more accurate when accounting for the decomposed set.

6.3 Experimental Setup

We run *SpinQ* on a laptop with an Intel Core i7-3610QM CPU @ 3.20 GHz and 16 GB DDR3 memory. *SpinQ* is written in Python 3.9.6 version.

7 EVALUATION AND ANALYSIS

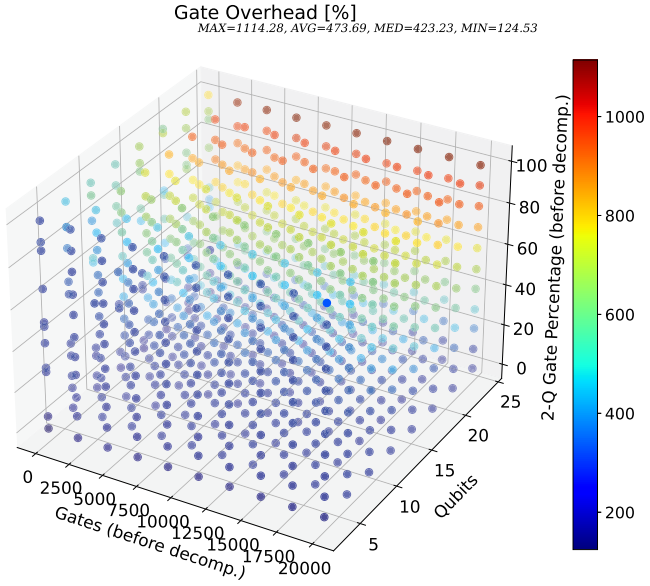
In this section, we present an in-depth performance analysis of *SpinQ* when mapping a broad range of quantum algorithms on the crossbar architecture. We then form architectural and mapping insights for each performance metric. More specifically, gate overhead and corresponding insights are presented in Sections 7.1 and 7.2, depth overhead in Sections 7.3 and 7.4, and ESP in Sections 7.5 and 7.6. Finally, we show results regarding the compilation time of *SpinQ* in Section 7.7 to asses its scalability capability.

7.1 Gate Overhead

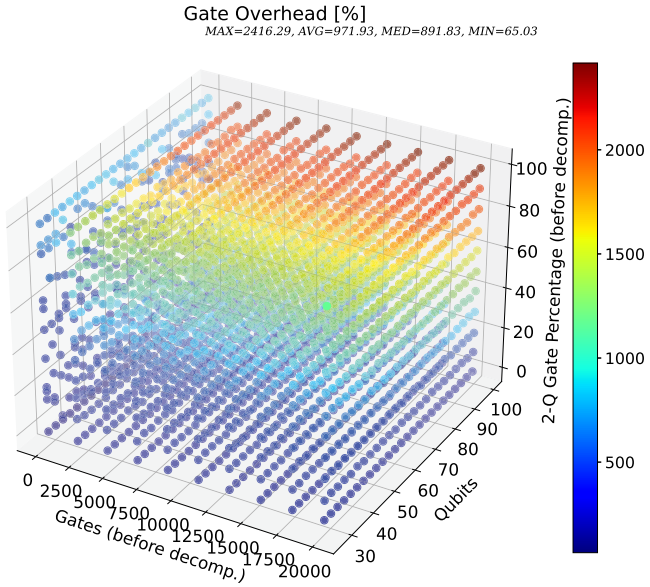
To start with, we analyze the gate overhead trend in a wide range of quantum algorithms. In Figure 8, we have mapped random uniform circuits on the crossbar architecture. Focusing on Figure 8(a), which reaches up to 25 qubits, we observe that as we go from low to high number of qubits and from low to high percentage of two-qubit gates, the gate overhead increases (from blue to red). More precisely, higher qubit counts imply larger crossbar topologies and thus potentially longer routing distances, i.e., more shuttle-based SWAPs. Furthermore, higher percentages of two-qubit gates potentially lead to more routing of qubits. These observations verify that the main source of gate overhead is indeed the routing of qubits for two-qubit gates (see Section 5.1). We also notice that the number of gates has a small but noticeable influence on the gate overhead. To further observe the trend when increasing the number of qubits, we changed the range of qubits from [3–25] to [25–99] in Figure 8(b). We see once more that the gate overhead increases as we go from low to high number of qubits and percentage of two-qubit gates. As expected, the gate overhead, shown on the color bars, of the [25–99] qubit range is on average 102.49% higher than that of the [3–25] qubit range because of the increased routing distances.

So far, the above random algorithms were generated to have control of different circuit parameters (i.e., number of qubits and gates and two-qubit gate percentage) in a way to broadly cover the parameter space and up to certain boundaries. However, they might not be representative of real algorithms from a circuit structure point of view (e.g., how two-qubit gates are distributed among qubits or the degree of operation parallelism). Therefore, we then mapped real algorithms from the RevLib [65] and Qlib [33] libraries, Grover and QFT resulting in the gate overhead shown in Figures 9, 10, and 11. In Figure 9(a) we can observe that benchmarks "cluster" together in similar colors, namely shades of blue, green, yellow, and red. This implies that similar benchmarks, meaning with similar parameters and structure, have similar gate overhead. Note that whereas random uniform algorithms have the same circuit structure because of the way they are generated, RevLib algorithms present different structural parameters not only compared to the randomly generated circuits but also between them. For this reason, correlations such as the higher the number of qubits and two-qubit gate percentage get, the higher the gate overhead, are not as evident as for the random circuits. Similarly, in Figure 9(b) we have executed Grover's and QFT algorithms. With these simulations, we also want to perform a scalability analysis of algorithms, which is not possible with RevLib circuits. From a first observation, it seems that QFT (top dots) produces higher gate overhead due to the higher two-qubit gate percentage compared to Grover's algorithm. However, once again, this cannot be conclusive, as they scale in different rates of benchmark characteristics.

To further analyze how structural circuit parameters impact the gate overhead, we mapped algorithms with similar rates of number of gates, qubits, percentage of two-qubit gates, and QIGs. First, note in Figure 10 that the Cuccaro Adder (top line in Figure 10) has a small drop in the percentage of two-qubit gates that goes from 71.43% to 66.75% when increasing in size (number of qubits), whereas the Vbe Adder (bottom line) maintains a lower percentage of 50% for the same qubit increase. One can immediately observe that the Cuccaro Adder shows a higher gate overhead

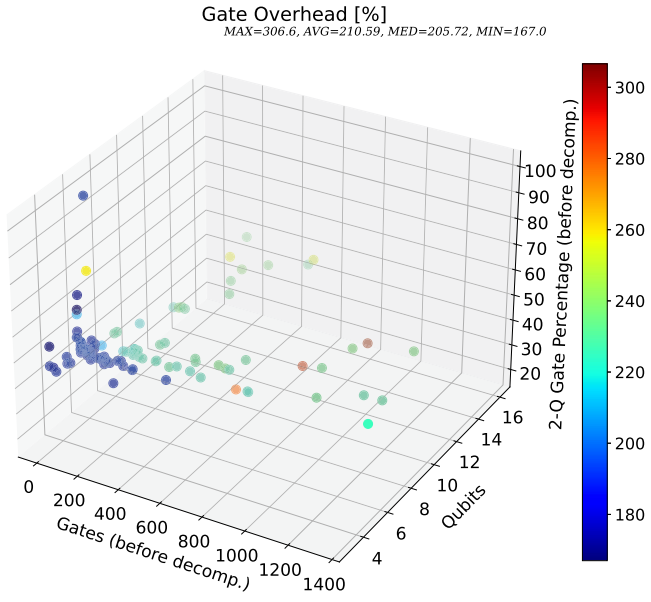


(a)

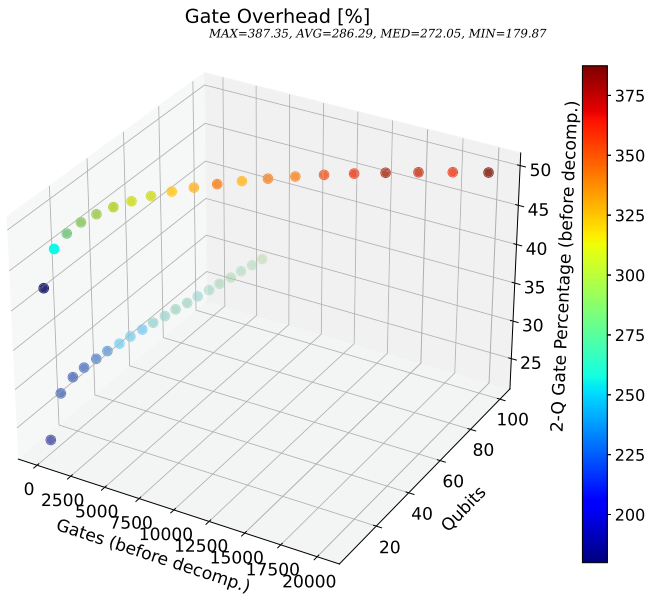


(b)

Fig. 8. Resulting gate overhead when 3,630 random uniform quantum algorithms are mapped onto the crossbar architecture. The three axes correspond to benchmark characteristics, namely, the number of gates [50–20,000], number of qubits [3–99] (split into two subfigures), and two-qubit gate percentage [0–100].



(a)



(b)

Fig. 9. (a) Resulting gate overhead when mapping quantum algorithms from the RevLib library onto the crossbar architecture. The three axes correspond to benchmark characteristics, namely, number of gates [5–1,400], number of qubits [3–16], and two-qubit gate percentage [18.75–100]. RevLib algorithms consist of reversible [3] quantum algorithms including, but not limited to, arithmetic and encoding functions [65]. (b) Resulting gate overhead when mapping Grover's (bottom line of data points) and QFT (top line of data points) quantum algorithms onto the crossbar architecture. The three axes correspond to benchmark characteristics, namely, number of gates [52–20,050], number of qubits [5–100], and two-qubit gate percentage [22.86–49.63].

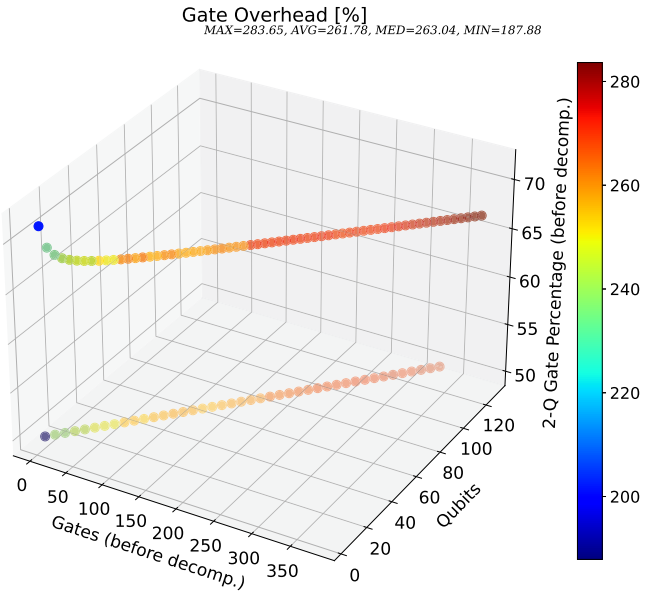


Fig. 10. Resulting gate overhead when mapping the Cuccaro Adder (top line of data points) and the Vbe Adder (bottom) quantum algorithms from the Qlib library onto the crossbar architecture. The three axes correspond to benchmark characteristics, namely, number of gates [4–385], number of qubits [4–130], and two-qubit gate percentage [50–71.43].

up to 284% due to the higher two-qubit gate percentage compared to the 271% of Vbe Adder, matching the conclusions made for Figure 8. However, as we emphasized above, in the case of real algorithms comparisons can only be properly made when looking not only at their circuit parameters but also at their more structural ones such as the QIG.

For this reason, in Figure 11 we show the derived QIGs from Vbe Adder's 40-qubit circuit, Cuccaro Adder's 38-qubit circuit, and Cuccaro Multiplier's 21-qubit circuit alongside their gate overhead in relation to the number of qubits and two-qubit gate percentage. In these QIGs, nodes correspond to qubits and edges to qubit interactions, i.e., two-qubit gates. The particular QIGs size selection was made to easily show their structure. We immediately observe similarities in the QIGs of the two Adders as the distribution of interactions is almost identical. More specifically, we see two to three interactions per qubit on average, with others close to their logical qubit number. Such a visual observation can be also quantified with the average QIG *degree*, which is calculated to be 3 for both. It is not surprising, therefore, that the higher gate overhead of Cuccaro Adder is indeed due to the higher percentage of two-qubit gates compared to Vbe Adder.

However, note that the Cuccaro Multiplier has the highest gate overhead of all three (309%) despite having a lower two-qubit gate percentage than the Cuccaro Adder. Looking at its much more connected QIG implies a denser qubit interaction distribution, compared to the others. Its average *degree* is determined to be 8; higher than that of the two other algorithms. Because of this, more routing is needed to connect nearly all qubits across the topology.

7.2 Insights from Gate Overhead Analysis

Accounting for the routing constraints, as discussed in Section 4, mapping on the crossbar architecture is not a trivial task. In fact, we have emphasized the importance of conceptualizing and developing new routing techniques that specifically can address the unique mapping chal-

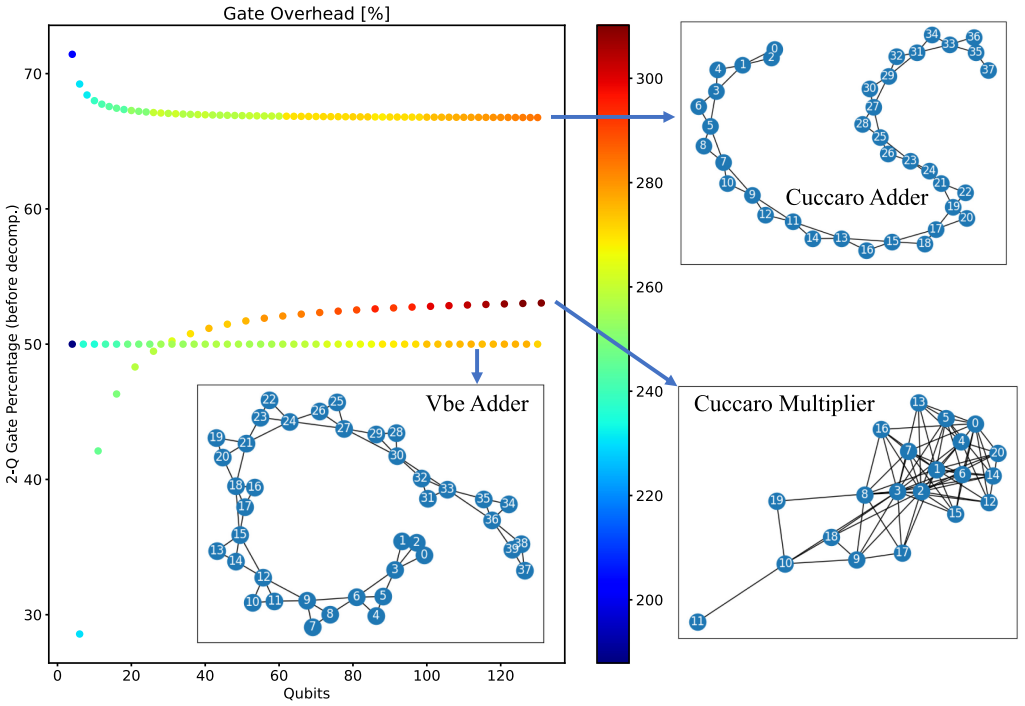


Fig. 11. Resulitng gate overhead when the Vbe Adder, Cuccaro Adder, and Cuccaro Multiplier from the Qlib library are mapped onto the crossbar architecture alongside their QIG consisting of 40, 38, and 21 qubits, respectively. The y-axis represents the two-qubit gate percentage and the x-axis the number of qubits. We see gate overhead to be influenced not only by the number of qubits and two-qubit gate percentage but also by the qubit interaction distribution.

lenges of spin-qubit architectures. More specifically, with the adoption of the checkerboard pattern combined with the shuttle-based SWAPs, we can provide a scalable solution of qubit routing for two-qubit gates. Additionally, the complexity only scales with the number of two-qubit gates, therefore being a viable solution for large-scale implementation. However, this technique makes two-qubit gate routing the highest source of gate overhead, and it can dramatically increase it with higher qubit counts and a higher percentage of two-qubit gates (see Figures 8 and 10). Moreover, in Figure 11 we saw that gate overhead can also be increased by a more connected QIG even though other circuit parameter values are comparatively lower. This shows the importance of basing circuit performance evaluation not only on simple circuit parameters but also on other "hidden" structural characteristics such as the qubit interaction distribution.

Having said that, the second biggest source of gate overhead originates from X or Y qubit rotations. This is due to the unprecedented semi-global rotation scheme. As mentioned in Section 4.2, this is the first time that single-qubit gate mapping requires additional routing instructions (i.e., produce gate overhead) compared to other qubit architectures. In comparison, neutral-atom architecture [17, 30, 45, 51, 66] demonstrated small algorithms execution by constructing local R_ϕ rotations at an arbitrary angle or axis by synthesizing a local R_z in between two global R_{xy}. This is a similar concept to the semi-global rotation mapping scheme of the crossbar architecture, but it differs in two key aspects. First, one has the ability of global rotation, whereas the other is only semi-global. This could be an advantage or a disadvantage in terms of performance depending on the algorithm at hand and on the mapping techniques used. Second and more importantly, the

crossbar architecture demands a more constrained scheme that necessitates routing operations. These operations must be executed meticulously to address the distinct architectural restrictions, such as ensuring the availability of empty sites for qubit movement, adhering to the shuttling signal constraints, and diligently sidestepping any potential conflicts. Therefore, this feature is exclusive and raises unique mapping challenges for spin-qubit architectures and calls for careful considerations during the compilation process, as explained in Section 4.

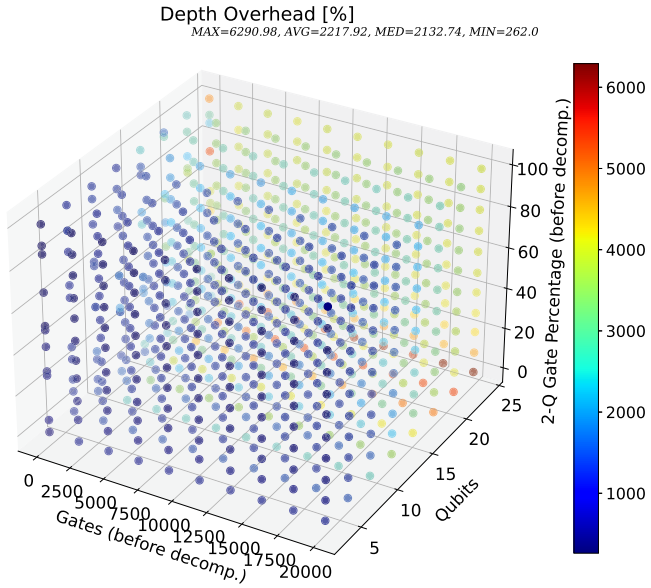
The previous two facts inspire novel mapping techniques for the crossbar architecture and potentially for other spin-qubit architectures with similar characteristics that can increase performance, namely

(1) Developing a routing solution dedicated to accounting for potential conflicts and constraints can reduce the gate overhead resulting from the shuttle-based SWAPs. Such a generalized routing algorithm could also include SWAP interactions (two consecutive $\sqrt{SWAP}$s) and *CPHASE* interactions. For instance, there can be scenarios that choosing a more noisy two-qubit interaction, for the purpose of avoiding an upcoming conflict, could result in higher ESP. Additionally, such a heuristic algorithm can allow multiple control or target qubits [29] to be shuttled around the topology enabling for parallelization of many two-qubit gates while avoiding high error variabilities in the topology [55]. However, such a solution must be implemented with the complexity in mind such that it will not make it unviable on large scale.
(2) A more efficient routing algorithm for single-qubit gates can significantly reduce the gate overhead, such that a specific rotation scheme to rotate targeted qubits is used less often. Such an algorithm can route qubits to the appropriate odd or even columns before the execution of single-qubit gates eliminating the need to apply any scheme afterward, such as the one in Section 4.2.
(3) Combining the previous two points, there can be a unified algorithm implementing both. In such an algorithm, upcoming routing for single-qubit gates is accounted for when routing for two-qubit gates and vice versa.
(4) Finally, an initial placement algorithm can take into account not only two-qubit gates but single-qubit gates as well. Since the positions of qubits influence the gate overhead resulting from single-qubit gate mapping (due to the semi-global rotation scheme), an extension of an initial placement algorithm accounting for single-qubit gates can further reduce the gate overhead.

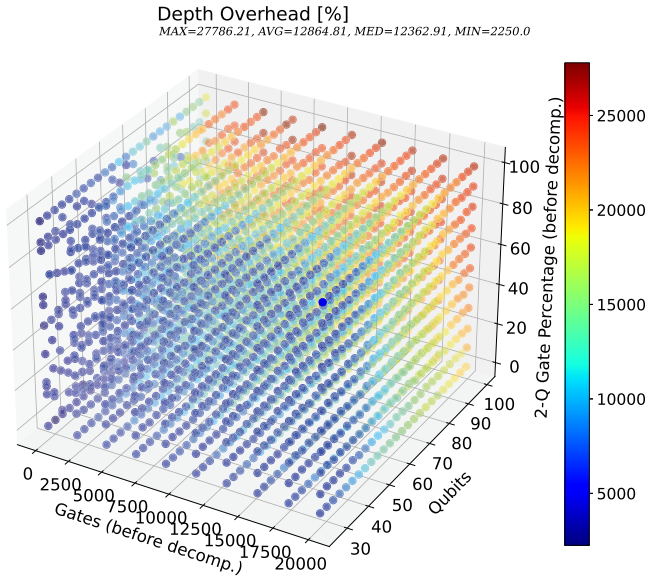
Last, we have emphasized that to concretely evaluate results, there has to be sufficient characterization of benchmarks, especially when evaluating novel architectures and mapping techniques. In our analysis, we did not rely only on simple benchmark parameters, such as the percentage of two-qubit gates, but also on the internal structure of benchmarks using the QIG.

7.3 Depth Overhead

This time, we analyze in Figure 12 the depth overhead when mapping onto the crossbar the same random uniform benchmark set as in Figure 8. It can be observed that the trend (colors) of the depth overhead changes for different ranges of number of qubits as shown in the two subfigures. Knowing that the main source of depth overhead originates from X or Y gates (at least three additional cycles), we expect the depth overhead to become higher in lower regions of two-qubit gate percentage. That is observed in Figure 12(a), where the number of qubits goes up to 25. However, moving on to Figure 12(b), we see that this trend changes. Now, due to the higher number of qubits, routing distances have increased, thus routing for two-qubit gates dominates the depth overhead.



(a)



(b)

Fig. 12. Resulting depth overhead when 3,630 random uniform quantum algorithms are mapped onto the crossbar architecture. The three axes correspond to benchmark characteristics, namely number of gates [50–20,000], number of qubits [3–99] (split into two subfigures), and two-qubit gate percentage [0%–100%].

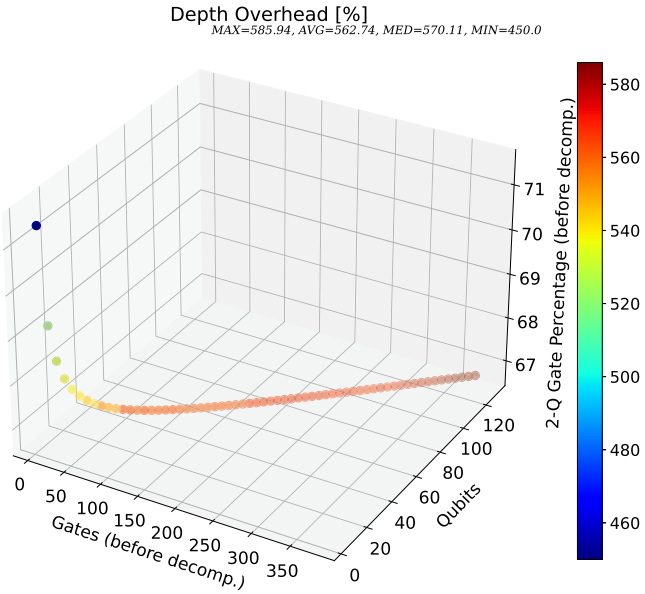


Fig. 13. Resulting depth overhead when Cuccaro Adder from the Qlib library is mapped onto the crossbar architecture. The three axes correspond to benchmark characteristics, namely number of gates [4–385], number of qubits [4–130], and two-qubit gate percentage [66.75–71.43].

This is apparent by its increase (from blue to red color) as we go from lower qubit counts to higher qubit counts and as we go from low to higher percentage of two-qubit gates. Finally, this fact is also apparent in the absolute values of depth overhead of the two subfigures. Note also that the number of gates has a slight influence on the depth overhead, but it is not as relevant as the other characteristics discussed above.

Moving on, Figure 13 shows the depth overhead of a Cuccaro Adder when scaling it up from 4 to 130 qubits. In the range of 4 to 20 qubits, we observe an increase in depth overhead as the percentage of two-qubit gates decreases, which aligns with the remarks about the main source of depth overhead (i.e., the X or Y gates). Then, for an increasing number of qubits (from 20 qubits on) and at an almost constant two-qubit gate percentage (67%), the depth overhead increases at a slower rate. Here we conclude, once again, that two-qubit gate routing starts to dominate the depth overhead as routing distances become larger.

In most previous works, the amount of two-qubit gates is the main circuit characteristic to anticipate how much qubit routing will be needed for a specific quantum algorithm and therefore the major and only source of gate/depth overhead. However, in the crossbar architecture, and potentially in other spin-qubit crossbar designs, single-qubit gates can also contribute to this overhead as discussed before. It is then important to have a closer look at the X or Y rotation gate percentage and further analyze how it impacts the depth overhead. Additionally, after the gate decomposition step, the percentages and ratios between all gate types are changed. To illustrate this, imagine a quantum circuit that originally consists of a low number of $CNOT$ gates and no Z gates. After the decomposition to gates supported by the crossbar architecture, the percentage of Z rotation gates will increase, and, consequently, the two-qubit gate percentage will decrease, as $CNOT$ gates are decomposed as $Ry(\frac{\pi}{2})$, two $\sqrt{SWAP}$, S, $S\dagger$, and $Ry(\frac{-\pi}{2})$. Thus, it is relevant to consider this gate percentage change in our analysis, as ultimately the executable circuit will only consist of native gates. To summarize, as overhead comes from mapping different types of gates on the crossbar,

individually distinguishing between them, in particular after decomposition, can increase the accuracy of our evaluations.

To illustrate the previous point, in Figure 14(a) we show the depth overhead of the Cuccaro Adder (upper dots) and the Vbe Adder (lower dots) with the same ranges as in Figure 10. Note that the y-axis corresponds to the percentage of X or Y rotation gates after decomposition. From this new perspective, we clearly see their difference in actual (i.e., executed by the architecture) X or Y rotation gate percentage. On average the depth overhead of the Vbe adder is 196% higher than the Cuccaro Adder for the same range of qubits. As explained before, the highest source of depth overhead comes from X or Y rotations gates, which explains the large depth overhead difference between those two algorithms.

In contrast, in Figure 14(b), we show the depth overhead after decomposition of the same algorithms with the same ranges as in Figure 9(b). This time, Grover's algorithm (top dots) shows on average 113% higher gate overhead than of QFT algorithm (lower dots). Note here that these two algorithms are plotted in relation to their Z gate percentage in the z-axis. Thus, their performance variations can be partially attributed to their differences in Z gate percentage, though this cannot be a definitive explanation. As previously stated, drawing a direct comparison is less straightforward due to disparities in algorithm structure and differing rates of benchmark characteristics.

7.4 Insights from Depth Overhead Analysis

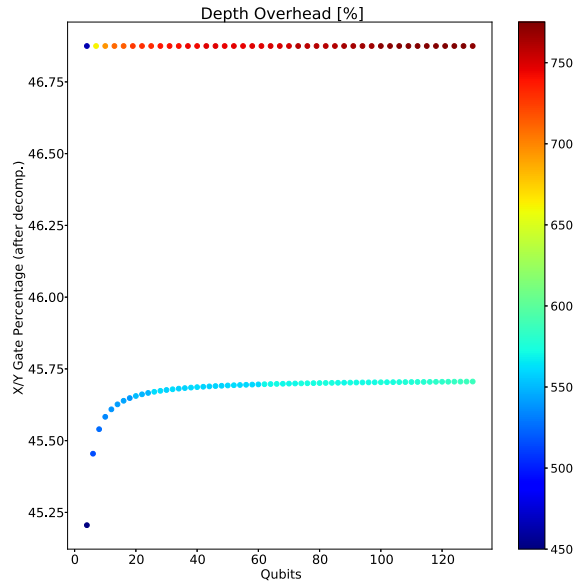
From the previous analysis, we can observe that trends can change based on the parameter ranges of benchmarks. This is because different sources of depth overhead contribute with different rates based on the number of qubits (i.e., crossbar size). More specifically, the overhead contribution resulting from mapping X/Y gates was higher up to a certain number of qubits after which was exceeded by the contribution rate of two-qubit gates. We saw that exceeding a threshold of more than 20 qubits increases the depth overhead at a steadier pace, which specifically favored scalability for Cuccaro Adder in Figures 13 and 14. It is expected, however, that with different algorithms, there will be different trends. With such observations, we stress the importance of distinguishing between all gate types and especially after decomposition to better understand the performance impact of mapping. With that knowledge, we can create better mapping techniques and/or make an informed selection of algorithms to execute.

As mentioned before, the fact that gate overhead and routing can result from mapping single-qubit gates is unprecedented. Furthermore, we notice that mapping both single- and two-qubit gates requires additional shuttles, and they produce the highest gate and depth overhead. Therefore, novel mapping techniques minimizing all qubit movements (shuttles) can increase performance substantially, such as the ones discussed in Section 7.2. From an architectural point of view, since the shuttle operation is so relevant, there have to be as few operational constraints as possible when mapping them.

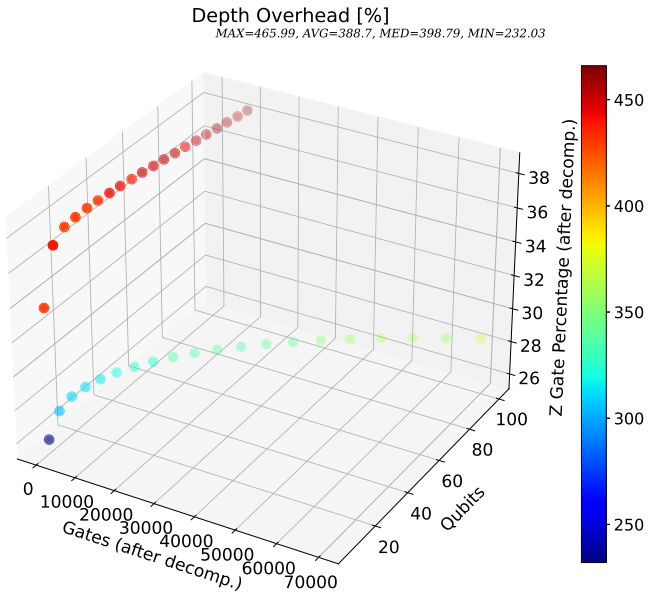
Finally, the current SpinQ version does not parallelize gates that are shuttle based. These are the resulting shuttle gates from the shuttle-based SWAP, mapping of single-qubit gates, and the two shuttling operations to facilitate a Z rotation. An improved version can involve a constraint and conflict check for any shuttle-based type gate to reach the full parallelization potential of the second pass, without increasing the time complexity.

7.5 Estimated Success Probability

In this section, we will show how the success probability of an algorithm drops after mapping it to the crossbar architecture. Before we continue, we have to mention that even with operational fidelities as high as 99.99% for single-qubit gates and shuttles (as suggested in Reference [32]) and 99.98% for $\sqrt{SWAP}$s, the ESP drops drastically to 0 in most algorithms with a high number of gates.



(a)



(b)

Fig. 14. (a) Resulting depth overhead when Cuccaro Adder (bottom line of data points) and Vbe Adder (top) from the Qlib library are mapped onto the crossbar architecture. The y-axis represents the X or Y gate percentage after decomposition, and the x-axis the number of qubits. (b) Resulting depth overhead when mapping Grover's (top line of data points) and QFT (bottom line of data points) quantum algorithms onto the crossbar architecture. The three axes correspond to benchmark characteristics after decomposition, namely number of gates [52–20,050], number of qubits [5–100], and Z gate percentage [25.97–38.29].

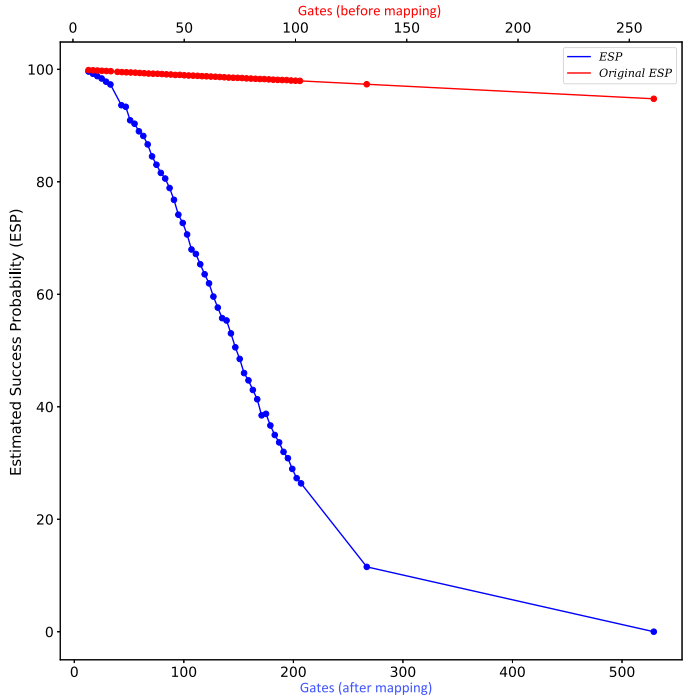


Fig. 15. ESP before and after mapping of Bernstein–Vazirani algorithm from 2 to 129 qubits.

For that reason, we only focused on the Bernstein–Vazirani algorithm, as it has a low percentage of two-qubit gates and therefore errors are mostly introduced by single-qubit gates.

Figure 15 shows the ESP of the Bernstein–Vazirani algorithm when scaling it from 2 to 129 qubits. The red line "Original ESP" refers to the ESP before mapping, and the blue line "ESP" refers to ESP after mapping. We observe a sharp ESP decrease approaching 10% for 267 gates after mapping with a slope rate of −0.6, which is caused by the increased number of gates. For 529 gates after mapping we obtained a 0% ESP. Another reason for the ESP decrease is the semi-global single-qubit rotation; for each of the X or Y gates contained in the circuit after decomposition, all qubits in odd or even columns are rotated even the ones that are not targeted for rotation. This is further explained in Section 4.2.

7.6 Insights from Estimated Success Probability Analysis

We observed a rapid decline in ESP in a minimally connected algorithm (mostly X or Y rotation gates), even though our equation did not include decoherence-induced errors [19, 26]. Although simple, Equation (2) is approximating a worse-case-scenario algorithm success rate. The main reason for this decrease is the resulting overhead when implementing single-qubit gates on specific qubits given the semi-global rotation scheme. Note that in this case, all qubits in either column parities are rotated thus each contributing to this ESP drop. Therefore, it is essential to determine which algorithms could take advantage of the semi-global control and/or develop architecture-specific mapping techniques to minimize the need for a scheme, as suggested in Section 7.2.

There are other sources of noise on real NISQ quantum devices that impact algorithm execution. Fortunately, it is expected that processors will gradually become more robust with better fabrication tolerances and error-mitigation techniques that will enable quantum error correction

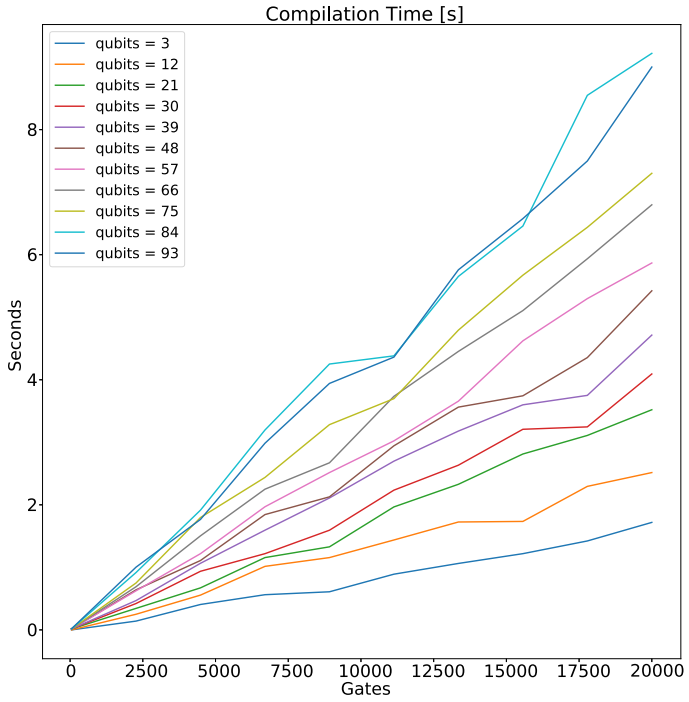


Fig. 16. Compilation time when mapping random uniform algorithms with 50% of two-qubit gates onto the crossbar architecture. We observe a linear relation that makes *SpinQ* suitable for scalable crossbar architectures.

protocols. It remains challenging, however, to accurately simulate errors in large-scale devices to derive the algorithm's success probability.

7.7 Compilation Time

Finally, we measure the compilation time of our solution to evaluate its scalability. The compilation time of *SpinQ's Integrated Strategy* can be seen in Figure 16 for a subset of the random uniform circuits that have been used in Figures 8 and 12. This subset consists of circuits with only 50% of two-qubit gates. With this subset we map the same number of gates for each gate type, thus all internal *SpinQ* processes are weighted equally. We observe a linear increase ($O(n)$) in compilation time in relation to the number of gates for each qubit count. This implies that our strategy is suited for scalable spin-qubit crossbar architectures. Improvements in compilation strategies of *SpinQ* can be directed toward reducing the slopes for each qubit count. This is further discussed in Section 8.1.

8 DISCUSSION AND FUTURE DIRECTIONS

8.1 Integrated Strategy Improvements

This is the first version of *SpinQ* and the *Integrated Strategy* is not utilized completely. More specifically, in this implementation, we are only parallelizing X or Y rotation gates and shuttles from the shuttle-based SWAP technique. However, there can be a few extensions to the *Integrated Strategy* that can provide better performance (less overhead and higher ESP). These improvements can be divided into two categories: (a) improvements that increase complexity marginally and (b) improvements that will increase complexity substantially. It is important to make this differentiation

because on a large scale we have to consider the tradeoff between complexity manifested as higher computation time as sizes increase and performance manifested as less overhead and higher ESP.

Improvements in category (a) are for the first and second pass of the *Integrated Strategy* and will improve scheduling of gates that use the *QL* lines (i.e., shuttling, two-qubit gates, and Z rotations). Note that the *Integrated Strategy* is made to support such an improvement within the two passes and it will not increase its time complexity. In particular, the first pass could parallelize ideally Z rotations together with one two-qubit gate, in lines 13 and 21, respectively. Then in the second pass in lines 35 and 37 a similar conflict checking process can be followed as line 37 when completing the Z rotation mapping and schedule them in the least cycles possible. Furthermore, two-qubit gate routing could be better parallelized in the second pass instead of the pairing presented in Figure 7. Two-qubit gate ideal parallelization in the first pass constitutes routing in the second pass that will require a new routing algorithm to handle multiple gates at the same time. Thus, such an improvement belongs to category (b). Once again, each cycle remains dedicated to one gate type; therefore, fine-tuning pulse durations in real devices is still possible.

Moving on to the next category (b), it consists of all heuristic mapping algorithms (routing and initial placement) discussed in Sections 7.2, 7.4, and 7.6, which can be extended to other scalable spin-qubit architectures. This will enable complete parallelization of two-qubit gates and less routing for both, single- and two-qubit gates.

Finally, in Section 3.5 an ideal measurement process has been considered for this work. However, it would increase the accuracy of our performance evaluation to include a more realistic readout process in *SpinQ* as an additional step after the *Integrated Strategy*. To do that there needs to be a protocol to move data and ancilla qubits in an efficient manner during and after algorithm execution—a non-trivial task. An optimization algorithm will need to initialize ancilla qubits in the right places and time so they do not remain inactive for long periods. Then, optimize the measurement procedure to take as less steps and shuttling operations as possible. As a consequence, there will always be additional operational overhead and degradation of the algorithm's success during readout. Additionally, parallelization of qubit measurement is also a relatively unexplored topic, and it highly depends on the ever-developing hardware implementations of spin-qubits.

8.2 Strategy Comparisons

As we discussed in Section 4, the crossbar architecture comes with constraints that prevent full parallelization of quantum instructions. The crossbar, however, may reach two types of conflicts (i.e., unwanted interactions or blocked paths), even if all constraints are respected. For that reason, there must be some kind of compilation strategy between the scheduler and the router to prevent conflicts. In this work, we have implemented the *Integrated strategy*, which is different from the three strategies suggested in Reference [38]. Table 2 compares the computational complexity of these three strategies with our own. The *Backtrack* strategy suggested in Reference [38] avoids conflicts by trying alternative scheduling combinations. If after repeating this process the scheduler has backtracked to the first instruction of the cycle, meaning no more scheduling combinations, then a new routing path is generated by the routing algorithm, and the scheduling is repeated. This strategy can be quite complex as the worst-case scenario can un-route and unschedule all the gates going back to a completely un-mapped circuit until a conflict-free mapping is found. An improved version of this strategy, called *Suffer a side effect*, is a special case of the former and it is only preferred whenever a corresponding conflict can be corrected and if the correction is less costly than exclusively following the "backtracking" strategy. The final strategy, and the one implemented in Reference [38], is called *Avoid the deadlock*. This strategy, similarly

Table 2. Computational Complexity Comparison between Compilation Strategies for the Crossbar Architecture [32]

Strategy	Complexity
Backtrack [38]	$O(n^3)$
Suffer a side effect [38]	$O(n^2 log(n))$
Avoid the deadlock [38]	$O(n)$
Integrated (ours)	$O(n)$

With n we denote the number of gates in a quantum circuit.

to our *Integrated strategy*, is trying to avoid conflicts by parallelizing only X or Y gates. In this way, $\sqrt{SWAP}$s and shuttle operations cannot cause a conflict. However, in this strategy there is no synergy between the routing and scheduling stages as our *Integrated strategy* has, therefore there is little flexibility for improvements and performance cannot be easily improved while maintaining the same complexity. Our strategy is able to maintain the same $O(n)$ complexity even after improvements, particularly type (a) improvements referred to in Section 8.1

8.3 General Discussion

When developing novel mapping techniques for scalable quantum computing architectures such as the si-spin crossbar two main factors have to be considered: *scalability* and *adaptability*. As spin-qubit fabrication capabilities are improving, new architectural designs with potentially higher qubit counts will be explored. Therefore, from a computation/compilation time point of view, mapping techniques should be as scalable as the underlying technology. Practically, this implies that highly sophisticated and more complex mapping techniques might be excellent for a particular architecture and up to a certain number of qubits, but could be impractical for more qubits or even unusable for another architecture. In addition, as we are slowly exiting the NISQ era, quantum technologies will become more robust, especially with the use of quantum error correction techniques. Instead of primarily focusing on optimizing mapping techniques for specific hardware or algorithms, the priority could become the speed and adaptability of these techniques to accommodate a diverse array of quantum algorithms and larger qubit configurations.

9 CONCLUSION

Different quantum circuit mapping techniques have been developed to deal with the limitations that current quantum hardware presents and are being consistently improved to expand its computational capabilities by getting better and better algorithm success rates. The most advanced mapping methods focus on ion-trap and superconducting devices due to their "maturity" compared with other quantum technologies. However, spin-qubit-based processors have a great potential to scale rapidly and the first 2D crossbar architectures have been recently demonstrated. In this work, we focused on the quantum circuit mapping challenges of the newly emerging spin qubit technology for which highly specialized mapping techniques are needed to take advantage of its operational abilities. Specifically, we used the crossbar architecture as a stepping stone to explore novel mapping solutions while focusing on scalability. The crossbar architecture adopts a shared-control scheme, thus making it a great candidate to tackle the interconnect bottleneck. On that note, we have developed *SpinQ*, the first native compilation framework for spin-qubit architecture that we used to analyze the performance of synthetic and real quantum algorithms on the crossbar architecture. Through our analysis, we tried to inspire novel algorithm- and hardware-specific mapping techniques that can increase the performance while taking into account

compilation scalability. We also emphasized the importance of characterizing benchmarks before and after decomposition together with their QIG structure to better evaluate results. We plan to make *SpinQ* publicly available in the future.

REFERENCES

[1] Carmen G. Almudever, Lingling Lao, Xiang Fu, Nader Khammassi, Imran Ashraf, Dan Iorga, Savvas Varsamopoulos, Christopher Eichler, Andreas Wallraff, Lotte Geck, et al. 2017. The engineering challenges in quantum computing. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'17)*. IEEE, 836–845.

[2] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (2019), 505–510.

[3] Medina Bandić, Carmen G. Almudever, and Sebastian Feld. 2022. Interaction graph-based profiling of quantum benchmarks for improving quantum circuit mapping techniques. arXiv preprint arXiv:2212.06640.

[4] Medina Bandic, Sebastian Feld, and Carmen G. Almudever. 2022. Full-stack quantum computing systems in the NISQ era: Algorithm-driven and hardware-aware compilation techniques. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[5] Medina Bandic, Hossein Zarein, Eduard Alarcon, and Carmen G. Almudever. 2020. On structured design space exploration for mapping of quantum algorithms. In *Proceedings of the XXXV Conference on Design of Circuits and Integrated Systems (DCIS'20)*. IEEE, 1–6.

[6] Francesco Borsoi, Nico W. Hendrickx, Valentin John, Sayr Motz, Floor van Riggelen, Amir Sammak, Sander L. de Snoo, Giordano Scappucci, and Menno Veldhorst. 2023. Shared control of a 16 semiconductor quantum dot crossbar array. *Nature Nanotechnology* (2023), 1–7.

[7] Jelmer M. Boter, Juan P. Dehollain, Jeroen P. G. Van Dijk, Yuanxing Xu, Toivo Hensgens, Richard Versluis, Henricus W. L. Naus, James S. Clarke, Menno Veldhorst, Fabio Sebastiano, et al. 2022. *Phys. Rev. Appl.* 18, 2 (2022), 024053.

[8] Jelmer M. Boter, Juan P. Dehollain, Jeroen P. G. van Dijk, Yuanxing Xu, Toivo Hensgens, Richard Versluis, Henricus W. L. Naus, James S. Clarke, Menno Veldhorst, Fabio Sebastiano, et al. 2022. Spiderweb array: a sparse spin-qubit array. *Physical Review Applied* 18, 2 (2022), 024053.

[9] Sergey Bravyi, Oliver Dial, Jay M. Gambetta, Darío Gil, and Zaira Nazario. 2022. The future of quantum computing with superconducting qubits. *J. Appl. Phys.* 132, 16 (2022), 160902.

[10] Colin D. Bruzewicz, John Chiaverini, Robert McConnell, and Jeremy M. Sage. 2019. Trapped-ion quantum computing: Progress and challenges. *Appl. Phys. Rev.* 6, 2 (2019), 021314.

[11] Lukas Burgholzer, Hartwig Bauer, and Robert Wille. 2021. Hybrid schrödinger-feynman simulation of quantum circuits with decision diagrams. In *Proceedings of the IEEE International Conference on Quantum Computing and Engineering (QCE'21)*. IEEE, 199–206.

[12] Leon C. Camenzind, Simon Geyer, Andreas Fuhrer, Richard J. Warburton, Dominik M. Zumbühl, and Andreas V. Kuhlmann. 2022. A hole spin qubit in a fin field-effect transistor above 4 kelvin. *Nat. Electr.* 5, 3 (2022), 178–183.

[13] Anasua Chatterjee, Paul Stevenson, Silvano De Franceschi, Andrea Morello, Nathalie P. de Leon, and Ferdinand Kuemmeth. 2021. Semiconductor qubits in practice. *Nat. Rev. Phys.* 3, 3 (2021), 157–177.

[14] David P. Franke, James S. Clarke, Lieven M. K. Vandersypen, and Menno Veldhorst. 2019. Rent's rule and extensibility in quantum computing. *Microprocess. Microsyst.* 67 (2019), 1–7.

[15] Takafumi Fujita, Timothy Alexander Baart, Christian Reichl, Werner Wegscheider, and Lieven Mark Koenraad Vandersypen. 2017. Coherent shuttle of electron-spin states. *npj Quant. Inf.* 3, 1 (2017), 1–6.

[16] Craig Gidney and Martin Ekerå. 2021. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum* 5 (2021), 433.

[17] T. M. Graham, Y. Song, J. Scott, C. Poole, L. Phuttitarn, K. Jooya, P. Eichler, X. Jiang, A. Marra, B. Grinkemeyer, et al. 2021. Demonstration of multi-qubit entanglement and algorithms on a programmable neutral atom quantum computer. *arXiv:2112.14589*. Retrieved from https://arxiv.org/abs/2112.14589

[18] Ronald Hanson, Leo P. Kouwenhoven, Jason R. Petta, Seigo Tarucha, and Lieven M. K. Vandersypen. 2007. Spins in few-electron quantum dots. *Rev. Mod. Phys.* 79, 4 (2007), 1217.

[19] Jonas Helsen, Mark Steudtner, Menno Veldhorst, and Stephanie Wehner. 2018. Quantum error correction in crossbar architectures. *Quant. Sci. Technol.* 3, 3 (2018), 035005.

[20] N. W. Hendrickx. 2021. Qubit arrays in germanium.

[21] Nico W. Hendrickx, William I. L. Lawrie, Maximilian Russ, Floor van Riggelen, Sander L. de Snoo, Raymond N. Schouten, Amir Sammak, Giordano Scappucci, and Menno Veldhorst. 2021. A four-qubit germanium quantum processor. *Nature* 591, 7851 (2021), 580–585.

[22] Steven Herbert and Akash Sengupta. 2018. Using reinforcement learning to find efficient qubit routing policies for deployment in near-term quantum computers. *arXiv:1812.11619* (2018).
[23] Charles D. Hill, Eldad Peretz, Samuel J. Hile, Matthew G. House, Martin Fuechsle, Sven Rogge, Michelle Y. Simmons, and Lloyd C. L. Hollenberg. 2015. A surface code quantum computer in silicon. *Sci. Adv.* 1, 9 (2015), e1500707.
[24] Hsin-Yuan Huang, Michael Broughton, Jordan Cotler, Sitan Chen, Jerry Li, Masoud Mohseni, Hartmut Neven, Ryan Babbush, Richard Kueng, John Preskill, et al. 2022. Quantum advantage in learning from experiments. *Science* 376, 6598 (2022), 1182–1186.
[25] IBM. 2022. Qiskit Aer Library. Retrieved from https://qiskit.org/documentation/apidoc/aer_library.html
[26] Y. Kharkov, A. Ivanova, E. Mikhantiev, and A. Kotelnikov. 2022. Arline benchmarks: Automated benchmarking platform for quantum compilers. *arXiv:2202.14025*. Retrieved from https://arxiv.org/abs/2202.14025
[27] Thaddeus D. Ladd, Fedor Jelezko, Raymond Laflamme, Yasunobu Nakamura, Christopher Monroe, and Jeremy Lloyd O'Brien. 2010. Quantum computers. *Nature* 464, 7285 (2010), 45–53.
[28] Lingling Lao and Dan E. Browne. 2022. 2qan: A quantum compiler for 2-local qubit hamiltonian simulation algorithms. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 351–365.
[29] Lingling Lao, Hans van Someren, Imran Ashraf, and Carmen G. Almudever. 2021. Timing and resource-aware mapping of quantum circuits to superconducting processors. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* (2021).
[30] Harry Levine, Alexander Keesling, Giulia Semeghini, Ahmed Omran, Tout T. Wang, Sepehr Ebadi, Hannes Bernien, Markus Greiner, Vladan Vuletić, Hannes Pichler, et al. 2019. Parallel implementation of high-fidelity multiqubit gates with neutral atoms. *Phys. Rev. Lett.* 123, 17 (2019), 170503.
[31] Gushu Li, Yufei Ding, and Yuan Xie. 2019. Tackling the qubit mapping problem for NISQ-era quantum devices. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*. 1001–1014.
[32] Ruoyu Li, Luca Petit, David P. Franke, Juan Pablo Dehollain, Jonas Helsen, Mark Steudtner, Nicole K. Thomas, Zachary R. Yoscovits, Kanwal J. Singh, Stephanie Wehner, et al. 2018. A crossbar network for silicon quantum dot qubits. *Sci. Adv.* 4, 7 (2018), eaar3960.
[33] Chia-Chun Lin, Amlan Chakrabarti, and Niraj K. Jha. 2014. Qlib: Quantum module library. *ACM J. Emerg. Technol. Comput. Syst.* 11, 1 (2014), 1–20.
[34] Daniel Loss and David P. DiVincenzo. 1998. Quantum computation with quantum dots. *Phys. Rev. A* 57, 1 (Jan. 1998), 120–126. DOI: https://doi.org/10.1103/PhysRevA.57.120
[35] Lars S. Madsen, Fabian Laudenbach, Mohsen Falamarzi Askarani, Fabien Rortais, Trevor Vincent, Jacob F. F. Bulmer, Filippo M. Miatto, Leonhard Neuhaus, Lukas G. Helt, Matthew J. Collins, et al. 2022. Quantum computational advantage with a programmable photonic processor. *Nature* 606, 7912 (2022), 75–81.
[36] Igor L. Markov, Aneeqa Fatima, Sergei V. Isakov, and Sergio Boixo. 2018. Quantum supremacy is both closer and farther than it appears. *arXiv:1807.10749*. Retrieved from https://arxiv.org/abs/1807.10749
[37] Marcel Meyer, Corentin Déprez, Timo R. van Abswoude, Ilja N. Meijer, Dingshan Liu, Chien-An Wang, Saurabh Karwal, Stefan Oosterhout, Francesco Borsoi, Amir Sammak, others. 2013. Electrical control of uniformity in quantum dot devices. *Nano Letters* 23, 7 (2013), 2522–2529.
[38] Alejandro Morais Tejerina. 2019. Mapping quantum algorithms in a crossbar architecture.
[39] Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret Martonosi. 2019. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*. 1015–1029.
[40] Prakash Murali, Dripto M. Debroy, Kenneth R. Brown, and Margaret Martonosi. 2020. Architecting noisy intermediate-scale trapped ion quantum computers. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA'20)*. IEEE, 529–542.
[41] Prakash Murali, Norbert Matthias Linke, Margaret Martonosi, Ali Javadi Abhari, Nhung Hong Nguyen, and Cinthia Huerta Alderete. 2019. Full-stack, real-system quantum computer studies: Architectural comparisons and design insights. In *Proceedings of the ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA'19)*. IEEE, 527–540.
[42] Shin Nishio, Yulu Pan, Takahiko Satoh, Hideharu Amano, and Rodney Van Meter. 2020. Extracting success from IBM's 20-qubit machines using error-aware compilation. *ACM J. Emerg. Technol. Comput. Syst.* 16, 3 (2020), 1–25.
[43] Feng Pan and Pan Zhang. 2022. Simulation of quantum circuits using the big-batch tensor network method. *Phys. Rev. Lett.* 128, 3 (2022), 030501.
[44] B. Paquelet Wuetz, P. L. Bavdaz, L. A. Yeoh, R. Schouten, H. Van Der Does, M. Tiggelman, D. Sabbagh, A. Sammak, Carmen G. Almudever, F. Sebastiano, et al. 2020. Multiplexed quantum transport using commercial off-the-shelf CMOS at sub-kelvin temperatures. *Npj Quant. Inf.* 6, 1 (2020), 1–8.

[45] Tirthak Patel, Daniel Silver, and Devesh Tiwari. 2022. Geyser: A compilation framework for quantum computing with neutral atoms. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 383–395.

[46] S. J. Pauka, K. Das, R. Kalra, A. Moini, Y. Yang, M. Trainer, A. Bousquet, C. Cantaloube, N. Dick, G. C. Gardner, et al. 2019. A cryogenic interface for controlling many qubits. *arXiv:1912.01299*. Retrieved from https://arxiv.org/abs/1912.01299

[47] Matteo G. Pozzi, Steven J. Herbert, Akash Sengupta, and Robert D. Mullins. 2020. Using reinforcement learning to perform qubit routing in quantum compilers. *arXiv:2007.15957*. Retrieved from https://arxiv.org/abs/2007.15957

[48] John Preskill. 2018. Quantum computing in the NISQ era and beyond. *Quantum* 2 (2018), 79.

[49] Nils Quetschlich, Lukas Burgholzer, and Robert Wille. 2022. Predicting good quantum circuit compilation options. *arXiv:2210.08027*. Retrieved from https://arxiv.org/abs/2210.08027

[50] Salonik Resch and Ulya R. Karpuzcu. 2019. Quantum computing: An overview across the system stack. *arXiv:1905.07240*. Retrieved from https://arxiv.org/abs/1905.07240

[51] Cheng Sheng, Xiaodong He, Peng Xu, Ruijun Guo, Kunpeng Wang, Zongyuan Xiong, Min Liu, Jin Wang, and Mingsheng Zhan. 2018. High-fidelity single-qubit gates on neutral atoms in a two-dimensional magic-intensity optical dipole trap array. *Phys. Rev. Lett.* 121, 24 (2018), 240501.

[52] Animesh Sinha, Utkarsh Azad, and Harjinder Singh. 2022. Qubit routing using graph neural network aided monte carlo tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 9935–9943.

[53] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. 2020. t| ket>: A retargetable compiler for NISQ devices. *Quant. Sci. Technol.* (2020).

[54] Matthew A. Steinberg, Sebastian Feld, Carmen G. Almudever, Michael Marthaler, and Jan-Michael Reiner. 2022. Topological-graph dependencies and scaling properties of a heuristic qubit-assignment algorithm. *IEEE Trans. Quant. Eng.* 3 (2022), 1–14. DOI: https://doi.org/10.1109/TQE.2022.3160015

[55] Swamit S. Tannu and Moinuddin K. Qureshi. 2019. Not all qubits are created equal: A case for variability-aware policies for NISQ-era quantum computers. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*. 987–999.

[56] Joseph Tindall, Matt Fishman, Miles Stoudenmire, and Dries Sels. 2023. Efficient tensor network simulation of IBM's kicked ising experiment. *arXiv:2306.14887*. Retrieved from https://arxiv.org/abs/2306.14887

[57] Teague Tomesh, Pranav Gokhale, Victory Omole, Gokul Subramanian Ravi, Kaitlin N. Smith, Joshua Viszlai, Xin-Chuan Wu, Nikos Hardavellas, Margaret R. Martonosi, and Frederic T. Chong. 2022. Supermarq: A scalable quantum benchmark suite. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA'22)*. IEEE, 587–603.

[58] Diogo Manuel Antunes Lopes Valada. 2020. Predicting the fidelity of quantum circuits search for better metrics for the qubit mapping problem.

[59] L. M. K. Vandersypen, H. Bluhm, J. S. Clarke, A. S. Dzurak, R. Ishihara, A. Morello, D. J. Reilly, L. R. Schreiber, and M. Veldhorst. 2017. Interfacing spin qubits in quantum dots and donors'hot, dense, and coherent. *Npj Quant. Inf.* 3, 1 (2017), 1–10.

[60] M. Veldhorst, H. G. J. Eenink, Chih-Hwan Yang, and Andrew S. Dzurak. 2017. Silicon CMOS architecture for a spin-based quantum computer. *Nat. Commun.* 8, 1 (2017), 1–8.

[61] M. Veldhorst, J. C. C. Hwang, C. H. Yang, A. W. Leenstra, Bob de Ronde, J. P. Dehollain, J. T. Muhonen, F. E. Hudson, Kohei M. Itoh, A. Morello, et al. 2014. An addressable quantum dot qubit with fault-tolerant control-fidelity. *Nat. Nanotechnol.* 9, 12 (2014), 981–985.

[62] Menno Veldhorst, C. H. Yang, J. C. C. Hwang, W. Huang, J. P. Dehollain, J. T. Muhonen, S. Simmons, A. Laucht, F. E. Hudson, Kohei M. Itoh, et al. 2015. A two-qubit logic gate in silicon. *Nature* 526, 7573 (2015), 410–414.

[63] T. F. Watson, S. G. J. Philips, Erika Kawakami, D. R. Ward, Pasquale Scarlino, Menno Veldhorst, D. E. Savage, M. G. Lagally, Mark Friesen, S. N. Coppersmith, et al. 2018. A programmable two-qubit quantum processor in silicon. *Nature* 555, 7698 (2018), 633–637.

[64] Robert Wille, Daniel Große, Lisa Teuber, Gerhard W. Dueck, and Rolf Drechsler. 2008. RevLib: An online resource for reversible functions and reversible circuits. In *Proceedings of the 38th International Symposium on Multiple Valued Logic (ISMVL'08)*. IEEE, 220–225.

[65] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. 2008. RevLib: An online resource for reversible functions and reversible circuits. In *Proceedings of the International Symposium on Multi-Valued Logic*. 220–225.

[66] T. Xia, M. Lichtman, K. Maller, A. W. Carr, M. J. Piotrowicz, L. Isenhower, and M. Saffman. 2015. Randomized benchmarking of single-qubit gates in a 2D array of neutral-atom qubits. *Phys. Rev. Lett.* 114, 10 (2015), 100503.

[67] Jun Yoneda, Kenta Takeda, Tomohiro Otsuka, Takashi Nakajima, Matthieu R. Delbecq, Giles Allison, Takumu Honda, Tetsuo Kodera, Shunri Oda, Yusuke Hoshi, et al. 2018. A quantum-dot spin qubit with coherence limited by charge noise and fidelity higher than 99.9%. *Nat. Nanotechnol.* 13, 2 (2018), 102–106.

[68] D. M. Zajac, T. M. Hazard, X. Mi, K. Wang, and Jason R. Petta. 2015. A reconfigurable gate architecture for Si/SiGe quantum dots. *Appl. Phys. Lett.* 106, 22 (2015), 223507.

[69] Alwin Zulehner, Alexandru Paler, and Robert Wille. 2018. An efficient methodology for mapping quantum circuits to the IBM QX architectures. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 38, 7 (2018), 1226–1236.

[70] Floris A. Zwanenburg, Andrew S. Dzurak, Andrea Morello, Michelle Y. Simmons, Lloyd C. L. Hollenberg, Gerhard Klimeck, Sven Rogge, Susan N. Coppersmith, and Mark A. Eriksson. 2013. Silicon quantum electronics. *Rev. Mod. Phys.* 85, 3 (Jul. 2013), 961–1019. DOI: https://doi.org/10.1103/RevModPhys.85.961

Received 4 April 2023; revised 1 August 2023; accepted 29 August 2023