SpinQ: Compilation strategies for scalable spin-qubit architectures

N. Paraskevopoulos[1,2], F. Sebastiano[1,2], C. G. Almudever[3], and S. Feld[1,2]
[1]*Quantum and Computer Engineering Department, Delft University of Technology, 2628 CD Delft, The Netherlands*
[2]*QuTech, Delft University of Technology, 2628 CJ Delft, The Netherlands and*
[3]*Computer Engineering Department, Technical University of Valencia, Camino de Vera, s/n, 46022 València, Spain*

In most qubit realizations, prototype devices are available and are already utilized in both industry and academic research. Despite being severely constrained, hardware- and algorithm-aware quantum circuit mapping techniques have been developed for enabling successful algorithm executions during the NISQ era, targeting mostly technologies with high qubit counts. Not so much attention has been paid to the implementation of compilation methods for quantum processors based on spin-qubits due to the scarce availability of current experimental devices and their small sizes. However, based on their high scalability potential and their rapid progress it is timely to start exploring quantum circuit mapping solutions for these spin-qubit devices. In this work, we discuss the unique mapping challenges of a scalable spin-qubit crossbar architecture with shared control [1] and introduce *SpinQ*, the first native compilation framework for scalable spin-qubit architectures that maps quantum algorithms on this crossbar architecture. At the core of *SpinQ* is the *Integrated Strategy* that addresses the unique operational constraints of the crossbar while considering compilation (execution time) scalability, having 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 technique improvements for the crossbar architecture that could increase algorithm success rates and potentially inspire further research on quantum circuit mapping techniques for other scalable spin-qubit architectures.

I. INTRODUCTION

The prospect of quantum computing advantage is steadily becoming a reality [2–4]. The community is anticipating further advances that will allow quantum computing systems to become practical and to reach computational advantage [5]. 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 [6], in which devices can only handle small-sized quantum circuits. This is due to limitations in the number of qubits and high operational errors, the latter causing rapid quantum information deterioration. Combined with even more hardware constraints, such as cross-talk and limited classical-control resources [7, 8], 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 [9–19] 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 [20–25]. Therefore, the scientific community is envisioning two-dimensional spin-qubit architectural proposals that could alleviate some of the major challenges towards scalability. Recently, a crossbar array [26] 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 paper, 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 [1]. 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 discuss and implement possible mapping solutions based on [27, 28] while improving those works from a scalability standpoint. We emphasize the importance of performing an extensive performance evaluation process of novel mapping techniques. Note, that this compilation framework will not only allow quantum algorithm executions on scalable spin qubit hardware but more importantly will also help to form insights on the behaviour and performance of this new breed of architectures and provide some design guidelines for future developments.

The main contributions of this paper are:

1. An in-depth analysis of mapping challenges in order to create novel mapping techniques for spin-qubit crossbar architectures.

2. *SpinQ* – The first native compilation framework dedicated to scalable spin-qubit architectures which utilizes a more scalable compilation strategy compared to pre-

vious 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.

4. Deriving algorithmic- and hardware-specific mapping insights for the crossbar architecture and other spin-qubit architectures.

The remainder of this paper is structured as follows: In Sec. II the current progress and challenges of scalable spin-qubit architectures are presented. In Sec. III the crossbar architecture is introduced as a potential candidate in scaling quantum devices in two dimensions, as well as its native operations. In Sec. IV we comprehensively analyse the unique challenges of mapping quantum algorithms on the crossbar architecture which require novel mapping techniques. Then, in Sec.V we introduce *SpinQ* – the first native compilation framework for scalable spin-qubit architectures. In Sec. VII 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 Sec. VIII we discuss potential improvements of our compilation strategy and we compare its computational complexity to previous proposals. Finally, we conclude our work in Sec. IX.

II. SPIN QUBITS AS A SCALABLE PLATFORM

To fulfil the promise [6] 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 [8, 29]. It still remains to be seen which qubit implementation technologies (e.g., superconducting, trapped ions, quantum dots, photonics, defect-based on nitrogen-vacancy diamond centres) will succeed in scaling up quantum computing systems with high-quality qubits [30, 31]. 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, long coherence times, and the ability to operate in super-kelvin temperatures [20–25].

Despite the advantages just mentioned, there are still several challenges today towards 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 [22]. Formally, the interconnect bottleneck is described by Rent's exponent [32], 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 with 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 can not 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 [1, 22, 33]. This makes it difficult to successfully control many qubits with the same control line, thus contributing to the wiring scheme challenge (i.e., the interconnect bottleneck).

There have been significant efforts [1, 22, 32, 34–38] 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 primarily has been focused on one-dimensional spin-qubit arrays of small sizes [22], which are not easily scalable. Recently, a 2×2 spin-qubit processor [39] and a 4×4 spin-qubit device based on a crossbar architecture [1, 26] with shared control has demonstrated the potential to scale spin-qubit devices in two dimensions. As the technology is advancing and further reducing Rent's exponent, there will be a need to effectively map quantum algorithms on two-dimensional devices such as the crossbar architecture which comes not only with limited qubit connectivity but also with a new set of constraints. Therefore, there is an opportunity to explore its mapping challenges and propose novel solutions.

However, the sample space of these proposals is sparse and lacks a detailed description of hardware constraints. In combination with a lack of available devices for testing, leads to a lack of a proper evaluation tool capable of benchmarking various quantum algorithms. Therefore, mapping techniques have not been studied as much as other qubit technologies such as superconducting and ion traps. It also remains unclear whether existing techniques could be applicable. Then, even if such techniques are realized they could be incompatible with existing quantum compilation frameworks 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 evaluate and compare various architectural proposals under relevant application categories.

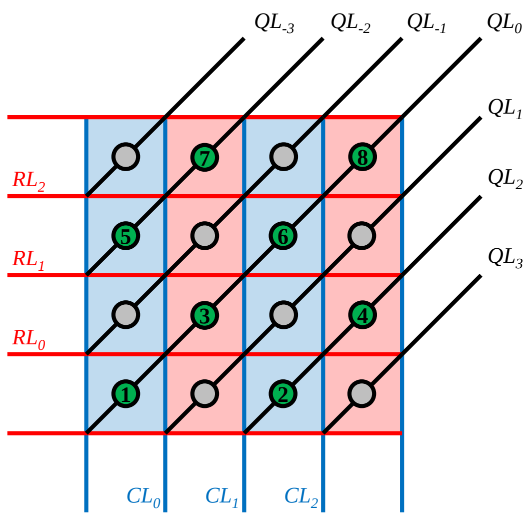


FIG. 1: Schematic overview of the crossbar architecture and operational control lines [1].

III. THE CROSSBAR ARCHITECTURE

The crossbar architecture for arranging spin qubits was introduced in [1] as a scalable solution to the interconnect bottleneck. Inspired by the crossbar architecture used in today's classical processors [1, 34], it adopts a similar characteristic, namely shared control. This leads to a quadratic reduction in control lines per qubit [28] and opens up the possibility for high integration of up to $1,000$ qubits in a single package. Qubits are defined by electron (or hole) spin states in Si-based quantum dots. In Figure 1, we illustrate a schematic overview of the crossbar architecture in which each site (circles) represents a quantum dot, some of which are occupied by spin-qubits (numbered, green circles). Spin qubits are usually sparsely initialized in a checker-board pattern to reduce potential cross-talk and to allow for long-range entanglement through shuttling qubits across the array [1]. Finally, the crossbar architecture requires high uniformity in the fabrication of materials to minimize operational errors. Fortunately, it is possible 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 to be duplicated in a network of modules. This can provide the means to realize quantum error correction (QEC) in large-scale systems enabled by fast-shuttling, low-error communication links. In this crossbar architecture, three different kinds of shared control lines are used to perform operations on the 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 Fig. 1 line QL_{-2} affects the sites in which spin-qubits 5 and 7 reside in. This imposes some restrictions in the parallelization of instructions, which we will discuss in Sec. IV. Below, we will abstractly describe the control properties for executing gates native to the crossbar architecture. A more detailed explanation is provided in [28]. **Two-qubit gates**. Two two-qubit gates $CPHASE$ and $\sqrt{SWAP}$ are supported by the crossbar, with the latter being chosen for this work due to its higher operational fidelity and faster execution time according to [1]. A $\sqrt{SWAP}$ can be performed when two qubits are vertically adjacent (i.e. same column) and the horizontal barrier between them is lowered. Then the QL lines going through the two qubits need to be in the same voltage potential for a specific duration of time to complete the $\sqrt{SWAP}$. **Qubit shuttling**. In the crossbar architecture, qubits can be moved around by performing shuttling operations. In this operation, the vertical and horizontal lines are used as barrier gates. Lowering or raising these barriers can create pathways from which qubits can move (shuttle) from one site to another with the use of DC signals through the diagonal lines. Fig. 2 shows an example of shuttling, in which spin-qubit 3 is moved one site to the left. Although this architecture can support gate-based communication with two subsequent $\sqrt{SWAP}$ gates as in 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-phase rotation which should be mitigated by timing such operations well ([1]). In the crossbar architecture, single-qubit gate rotations should be separated into two categories: Z-phase rotations and X or Y rotations. **Z-phase rotations**. Z-phase qubit rotations are controlled by a well-timed qubit shuttling to and from a neighbouring column [1, 27, 28]. This is due to the differences in Zeeman energies from column to column which imposes an alternating magnetic field on qubits, thus rotating them in the Z axis. When this shuttle is timed correctly, it rotates the qubit in the correct Z state. The diagonal qubit line provides the means to address multiple qubits, thus enabling parallel Z phase shifts across the topology. **X or Y rotations**. As for X or Y rotations, either all qubits belonging to red-coloured columns or all qubits in blue-coloured columns are rotated (see Fig. 1). This is called semi-global qubit rotation implemented by electron-spin-resonance ([40]). **Measurement**. The process of readout allows for local single qubit measurements based on the Pauli Spin Blockade (PSB) process [41]. With this process, the measurement outcome is determined by whether a qubit shuttle towards a horizontally adjacent ancilla qubit was successful or not.

IV. QUANTUM CIRCUIT MAPPING CHALLENGES OF THE CROSSBAR ARCHITECTURE

The quantum circuit mapping process 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. 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 which differ in strategy, methodology and performance metrics to optimize [9–19, 42].

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 single qubits, and the routing of qubits (i.e. moving qubits around).

1. *Parallelization of quantum operations*

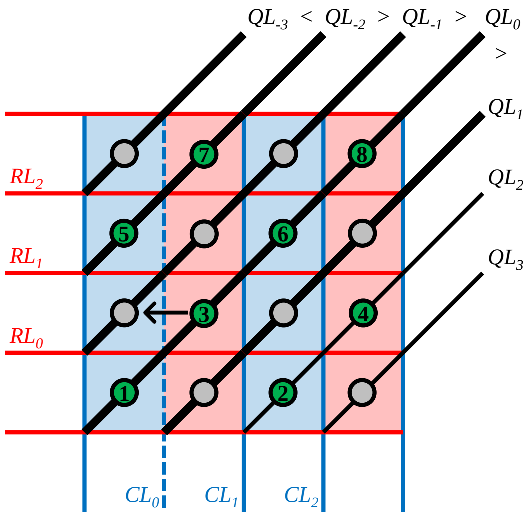


FIG. 2: Shuttling example of qubit 3 moved one site to the left, as the barrier CL_0 between origin and destination site is lowered and voltage of QL_{-1} is larger than $> QL_0$.

Most of the operation parallelization restrictions come from the fact that control lines are shared among multiple qubits, while each line has a specific role and relation to one another. It should also be noted that most operations must be implemented with strict pulse durations and time intervals depending on the site that gets addressed [1] due to fabrication imperfections [28]. Although such pulse durations have to be carefully considered in the mapping process by providing recent calibration data [13, 17, 18], 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 such considerations and can be added once calibration data are available.

To better illustrate what the conditions and constraints are

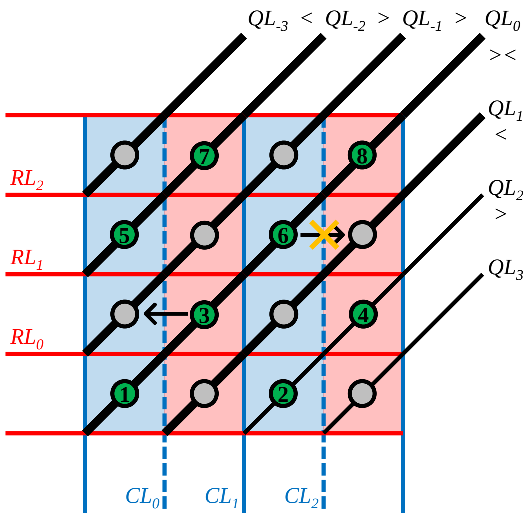


FIG. 3: Parallelizing shuttles of qubit 3 and 6 is not allowed due to violation of constraints.

when trying to parallelize quantum operations, let us consider the following example in which two shuttles are performed in parallel. As shown in Fig. 2, the following requirements must be fulfilled 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 Fig. 2.
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 Fig. 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 can not 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, we violate the fifth requirement of the first shuttle and, as a consequence, qubit 1 will shuttle to the right. Therefore, we can not shuttle qubits 3 and 6 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. Any violation of these conditions would potentially result in shuttling of unwanted qubits, in 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 or not. Unlike more popular qubit architectures based on superconducting or ion traps, this form of operational constraint is unique. On 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 (see Fig. 4a), because the former CL lines need to carry an alternating current (AC) signal while the latter require DC signals for raising or lowering the barriers.

2. *Application of X or Y rotations on single qubits*

As established in Sec. III, 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 of algorithm execution, not all qubits in odd or even columns should be rotated. Therefore, to compensate for unwanted X or Y rotations, one has to come up with a specific rotation scheme such that only the targeted qubits are rotated. In this work, we have implemented the scheme introduced by [28]. We illustrate how it works in Fig. 4, in which we are interested in rotating only qubit 5. This is another unique characteristic of this architecture, as additional gates are needed to perform single-qubit rotations on specific qubits, which impose new challenges to the mapping process.

3. *Routing of Qubits*

While we previously described the operational constraints to parallelize various gates in the crossbar architecture, we will now expand specifically on the qubit routing challenges.

Routing a qubit in the crossbar means that an electron (or a hole) is physically "pushed" to an empty site (i.e., an empty quantum dot). This mechanism is similar to a Quantum charged coupled device (QCCD) ion trap device when ions are shuttled through a common channel from trap to trap, assuming sufficient destination ion trap capacity [43]. 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.

Focusing on the crossbar, shuttling a qubit does not only depend on specific control signal requirements and available empty sites but on the positions of other qubits as well. We illustrate this fact with an example in Fig. 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 voltage relations to allow for correct shuttling. However, an unwanted interaction between two other qubits in the same row (qubits 2 and 4, circled) is concurrently caused, 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 [27, 28]. Lastly, 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, the same conflicts have to be considered and prevented when scheduling gates in parallel. For that, control signals and qubit positions must be carefully monitored within the mapping process. From the description above, it is clear that both, routing and scheduling processes, need to jointly work in a strategy to avoid conflicts and optimize for algorithm success rate. This will be part of *SpinQ*, presented in the following section.

V. *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 [27, 28] while improving them from a scalability standpoint.

Fig. 6 shows the schematic structure of our framework. As **input**, *SpinQ* accepts QASM format files that describe quantum circuits (used as benchmarks) in a device-independent manner. To increase flexibility, custom operations and their particular attributes can be defined in a hardware **architectural configuration** file. It can include operational attributes such as the duration of a gate, the mathematical description of the unitary matrices, associated gate fidelities, and architectural constraints, among others. Moving on, the **compiler** consists of a series of steps (called 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*. We did so not necessarily to maximize the performance of the algorithms when being executed on the crossbar, but rather to study how they behave on such architecture and focus on the scalability potential of spin-qubit technologies (see also Sec. VIII). The compiler's **output** is a QASM file which is compatible to be executed on the given crossbar architecture. Optionally, a **verification** step can take place to ensure the compiled

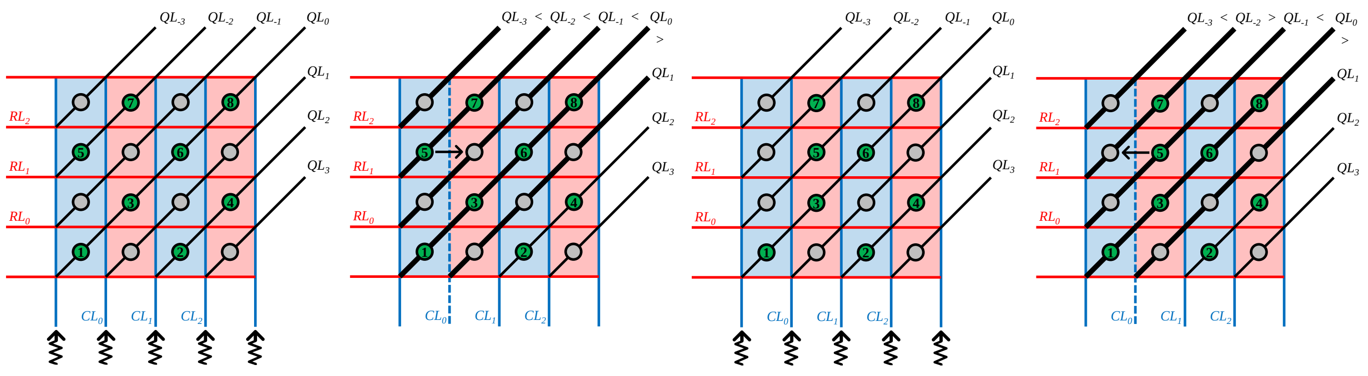

(a) Step 1 (b) Step 2 (c) Step 3 (d) Step 4

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, thus changing their state. The direction and frequency of these signals determine which columns (red or blue) and what rotation (X or Y gate) will be applied to the corresponding qubits. **(b) Step 2**: The targeted qubit 5 is moved with a shuttle operation to a different column parity. For this operation, the orthogonal lines (CL and RL) open and close as barriers and the diagonal lines (QL) create potential gradients to allow for qubit 5 to move (shuttle). Note that QL needs to have voltage relations with their neighbour QL lines. **(c) Step 3**: An inverse rotation is applied to 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.

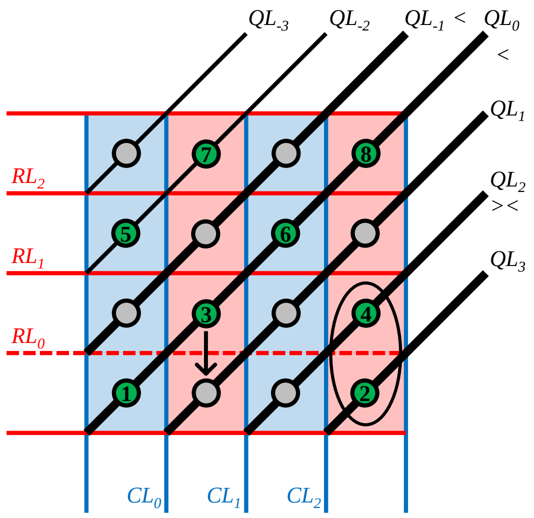


FIG. 5: Example of a conflict: the operational requirements of shuttling qubit 3 downwards have lowered the RL_0 barrier thus causing an unwanted interaction between qubits 2 and 4

circuit meets all operational constraints of the crossbar architecture without any conflicts. This step is implemented to be able to check the compatibility of architectural proposals that are not physically realized yet. 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 elements of the compiler.

A. Compilation steps

The compiler consists of the following steps:

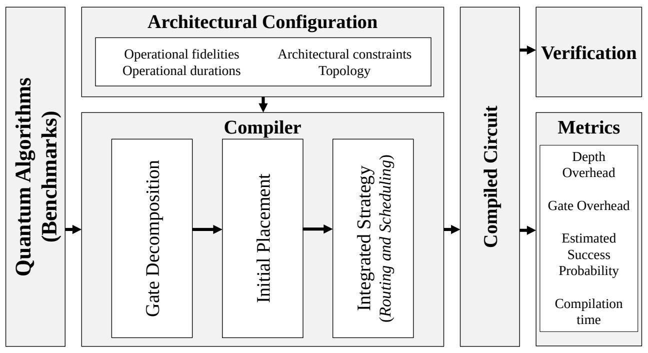


FIG. 6: Overview of our SpinQ framework proposed in this paper.

Decomposition of quantum gates. Inputted QASM quantum circuits are transformed into a custom-made intermediate representation (IR) data format. Quantum gates are then decomposed into gates native to the architecture based on the decomposition sequences specified in the architectural configuration file.

Physical initialization of spin qubits. A checkerboard pattern has been proposed [42] to allow space for qubits and ancilla qubits to move [27, 28]. The physical space achieved between the qubits not only facilitates shuttling qubits avoiding possible conflicts but also reduces crosstalk and enables surface code error correction [1]. As we will discuss later, maintaining this placement throughout a circuit execution plays an integral role in our compilation strategy. Having said that, initializing qubits in alternative patterns and changing them during execution is possible. This flexibility can be particularly advantageous to highly specialized mapping techniques for the crossbar as well as spin-qubit architectures in general.

Virtual-to-physical qubit initial placement. The current version of *SpinQ* associates virtual qubits of an algorithm with physical qubits (placed in the checkerboard pattern) in a one-to-one manner by numbering the physical qubits from left to right and from bottom to top (as shown in Fig. 1. In the results sections VII and VIII, 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).

Integrated Strategy for Routing and Scheduling. As explained in Sec. IV, both routing and scheduling techniques must avoid conflicts. To do that, a specific strategy needs to be followed. There can be various strategies for various goals with trade-offs between performance and compilation time. The presented *Integrated Strategy* tilts towards minimizing compilation time while having great prospects to be competitive against other solutions that focus on algorithm performance as will be discussed in Sec. VIII.

Firstly, in the *Integrated Strategy*, the checkerboard pattern qubit placement [1], also known as ”idle-configuration” in [28], should be maintained as much as possible. This provides at least two empty sites for every qubit to move towards to, at the beginning of each cycle. To maintain the checkerboard pattern throughout circuit execution when routing for two-qubit gates, a conflict-free shuttle-based SWAP technique can be used ([27]) as shown in Fig. 7. Note that this movement of qubits results in a gate overhead of 4 (i.e., 4 shuttle operations), but a depth overhead of 2, as these two shuttle pairs can always be executed in parallel. To bring the two qubits to the appropriate sites and allow two-qubit interactions, multiple shuttle-based SWAPs might be performed. For that, we have implemented a shortest-path algorithm based on the Manhattan distance. When one of the qubits is placed in the desired position, the next step is a horizontal shuttle, either to the left or to the right, after which the target and control qubits are vertically adjacent for interaction, and the checkerboard pattern is temporarily broken. Proceeding the $\sqrt{SWAP}$, a shuttle instruction 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.

So far, we have only talked about a routing technique for bringing together qubits for performing two-qubit gates. However, qubit routing is also needed for X or Y rotations to a specific qubit(s) and for shuttle-based Z rotations, as discussed in Sec. III. As a consequence, the ”idle configuration” should be maintained when routing for these gates as well. But once again, routing for single-qubit gates before the scheduling stage can be problematic as it can cause conflicts. For that reason, the second consideration of the *Integrated Strategy* is the integration of single-qubit gate routing within the scheduling stage, hence the name ”integrated”.

Then the *Integrated Strategy* continues with two passes. In the first pass, the scheduler tries to parallelize X or Y gates in an ideal manner and Z gates individually, ignoring any potential conflicts. This is no different than other single-qubit gate scheduling processes proposed for other qubit architectures. However, it differs on the second pass which 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 possibilities: (a) if no conflicts are detected when scheduling the necessary shuttle instructions required for each single-qubit gate, the new shuttle instructions are inserted between the current cycle and the next. (b) if conflict(s) are detected, the subset of the problematic gate(s) is removed and stored. Once the non-problematic gate subset is scheduled according to case (a), the problematic subset is recalled and iterated again. This time it constitutes a conflict-free cycle and is scheduled according to case (a). This way the second pass loops in total two times whenever there is a detected conflict.

Overall, the current implementation does not parallelize gates of different types in the same cycle, and thus each cycle is dedicated to one instruction type. Fortunately, the strategy described above and suggested extensions in Sec. VIII can be adapted to a real setup. As explained in Sec. IV, a fabricated crossbar device will most likely have material imperfections, thus requiring pulse calibration per site. As pointed out by [28, 44], 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 the cycle is possible before moving to the next.

B. 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.

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. 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. The main sources of gate overhead are the following:

- 4 additional shuttle instructions per shuttle-based SWAP for two-qubit gates
- At least 3 additional instructions for each X or Y rotation gate within the semi-global rotation scheme
- 2 additional shuttle instructions for each two-qubit gate
- 1 additional shuttle operation for each Z rotation gate

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

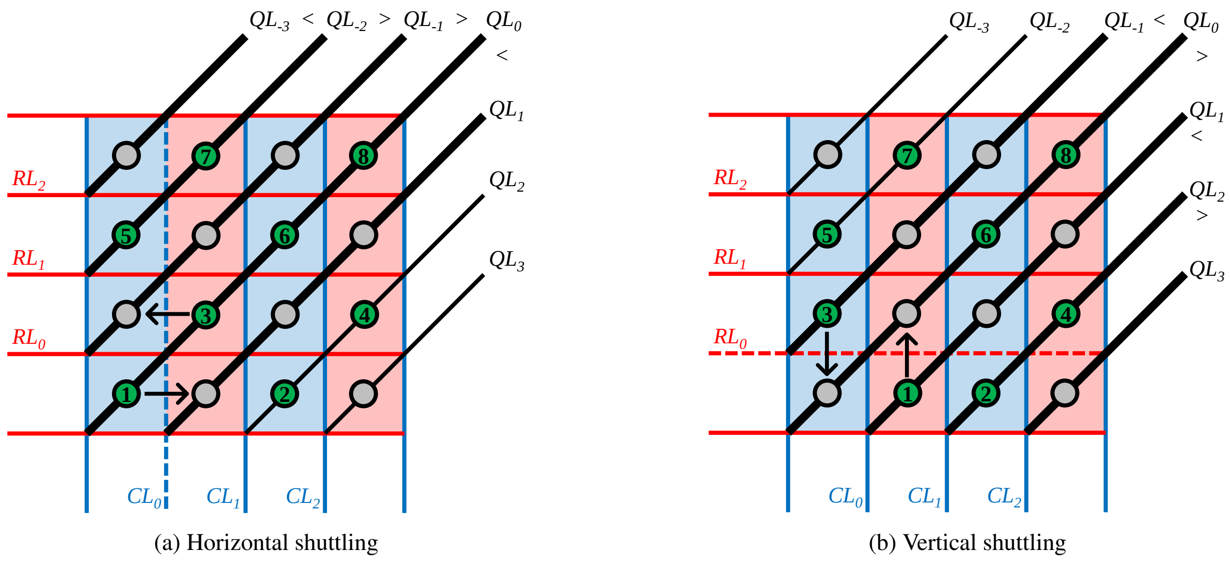


FIG. 7: Shuttle-based SWAP for two-qubit gate routing: With this technique, two diagonally neighbouring qubits exchange their position by consecutively performing two horizontal and two vertical shuttles.

equal to the minimum number of time steps of a circuit when executing gates in parallel [9, 10, 45–47]. We calculate depth overhead as the percentage relation of additional depth produced by the mapper to the circuit depth after decomposition. Note that the initial circuit depth is calculated after scheduling the circuit only by its gate dependencies, meaning without any architectural constraints. The main sources of depth overhead are:

- At least 3 additional cycles for each X or Y rotation gate due to the semi-global rotation scheme
- 2 additional cycles per shuttle-based SWAP for two-qubit gates
- 2 additional cycles for each two-qubit gate
- 1 additional cycle for each Z rotation gate

Estimated Success Probability. A key metric to assess the performance not only of the compiler but in general of a quantum computing system is the algorithm 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 using an approximation numerical method. One of the most commonly used methods to do so is considering the estimated success probability (ESP) of an algorithm [48]:

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

where i represents the ith time step and j the jth gate in the ith time step.

This method is far more efficient compared to using a Hamiltonian model. However, the accuracy of the estimation can be low due to its simplicity. 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 ($\sqrt{SWAP}$) 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 time step, j the jth gate in the ith time step and and x, y are the physical qubit(s) coordinates.

Compilation time. In this work, we are not only interested 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.

C. Verification

A verification tool is important to this work due to the lack of a working device for real-system testing. The tool is searching for mismatches between all shuttling sequences and the qubits position history stored during compilation. It also checks for conflicts, architectural constraint violations and state vector mismatches between and in each stage of the mapper. The latter uses the Qiskit Aer library [49].

VI. EXPERIMENTAL METHODOLOGY

Benchmarks. We have generated $3,630$ random uniform algorithms [50] 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 [51]. Quantum circuits from this library are often used in related quantum circuit compilation works [9, 11, 12] 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 [52] which contains real quantum algorithms in increasing size.

Benchmarks characterization. When it comes to performance evaluation, it is important to not only consider properties of the crossbar architecture but also the characteristics of quantum circuits. The simplest and most commonly [14] 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. Firstly, two benchmarks, for instance, could have the same parameter values but heavily differ in the circuit's structure [14]. When one of them has all pairs of qubits interact with each other will require more routing than the other which might have the same number of interactions, but with only one pair of qubits 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 distil its properties [14]. Having said that, we analyze QIGs visually only, as this is still an active field of research [14]. Despite that, we can nonetheless make concrete conclusions and form insights, making visual QIG assessments a viable tool to characterize algorithms. 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 actual executable set, meaning that evaluations can become more accurate when accounting for the decomposed set.

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

VII. EVALUATION AND ANALYSIS

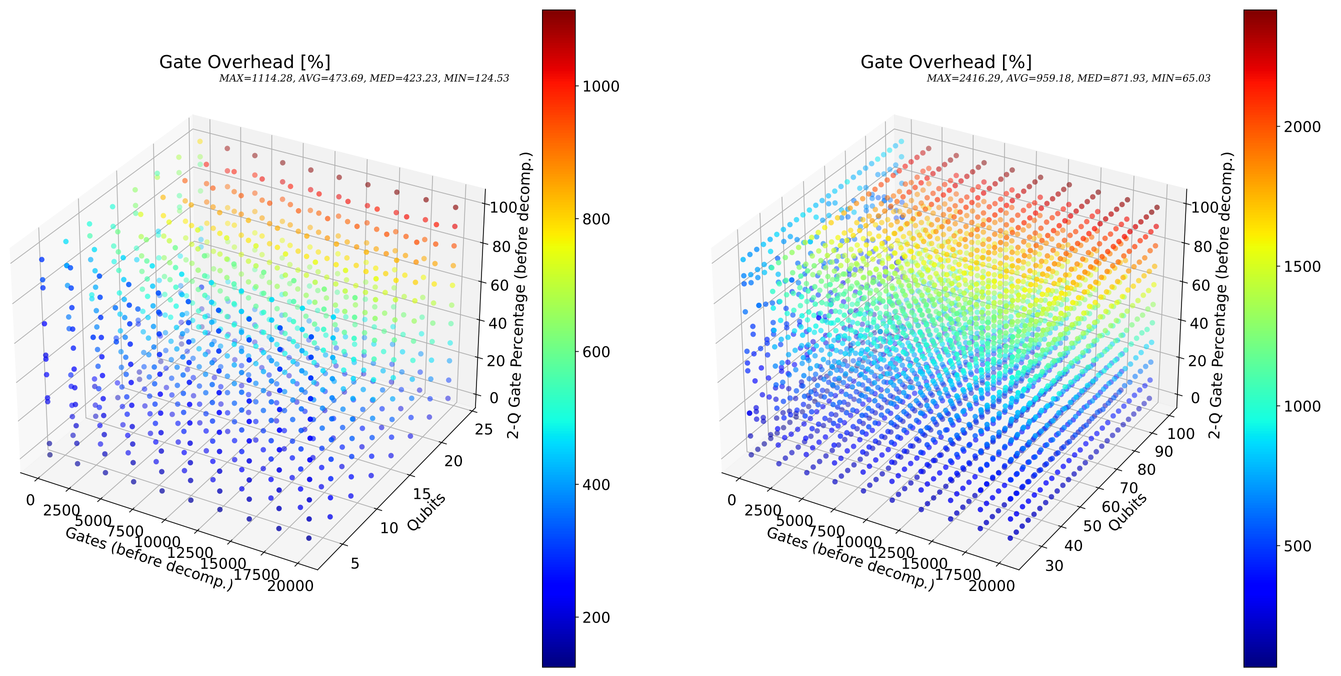
In this Section, we present an in-depth performance analysis of *SpinQ* when mapping a broad range of quantum algorithms on the crossbar architecture. We then form architectural and mapping insights for each performance metric. More specifically, gate overhead and corresponding insights are presented in Sec. VII A and VII B, depth overhead in Sec. VII C and VII D, and ESP in Sec. VII E and VII F. Finally, we show results regarding compilation time of *SpinQ* in Sec. VII G to asses its scalability capability.

A. Gate Overhead

To start with, we analyse the gate overhead trend in a wide range of quantum algorithms. In Fig. 8 we have mapped random uniform circuits on the crossbar architecture. Focusing on Fig. 8a, 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 color). More precisely, higher qubit counts imply larger crossbar topologies, 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 Sec. V A). 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 Fig. 8b. 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 and Qlib libraries resulting in the gate overhead shown in Fig. 9, Fig. 10, and Fig. 11. In Fig. 9 we can observe that benchmarks "cluster" together in similar colours, 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 all 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 percentage of two-qubit gates gets, the higher the gate overhead will be, are not as evident as before (i.e. for random circuits).

To further analyse how structural circuit parameters impact the gate overhead, we mapped algorithms with similar number of gates, qubits, percentage of two-qubit gates and QIG from



(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].

the Qlib library onto the crossbar architecture (see Fig. 10). With these simulations, we also want to perform a scalability analysis of the algorithms which is not possible with RevLib circuits. First, note that the Cuccaro Adder (top line in Fig. 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 increase in size. One can immediately observe that the Cuccaro Adder shows a higher gate overhead up to 284% due to the higher two-qubit gate percentage compared to the 271% of Vbe Adder, matching the conclusions made in Fig. 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 Fig. 11 we show the derived QIGs from Vbe Adders' 40-qubit circuit, Cuccaro Adders' 38-qubit circuit and Cuccaro Multipliers' 21-qubit circuit alongside their gate overhead in relation to the number of qubits and percentage of two-qubit gates. In these QIGs, nodes correspond to qubits and edges to qubit interactions, i.e., two-qubit gates. The particular size selection of these QIGs 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 2 to 3 interactions per qubit on average, with others close to their logical qubit number. Therefore, we can conclude that the higher gate overhead of Cuccaro Adder is 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. The reason behind this is the difference in its QIG, which is much more connected implying a denser qubit interaction distribution compared to the others. Because of this, more routing is needed to connect (nearly) all qubits across the entire topology.

B. Insights from gate overhead analysis

Accounting for the routing constraints, as discussed in Sec. IV, 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 challenges 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 Fig. 8 and 10). Moreover, in Fig. 11 we saw that gate overhead can also be increased by a more connected QIG even if other circuit parameter values are comparatively lower. This shows

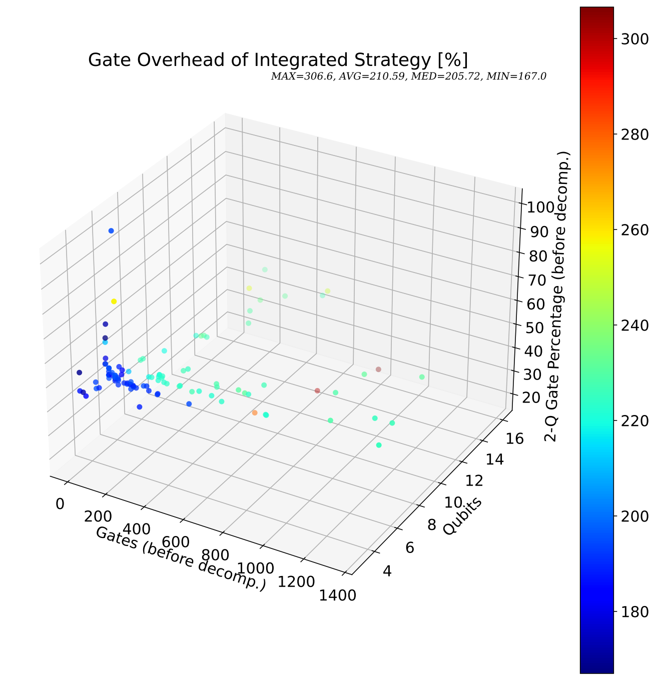


FIG. 9: Resulting gate overhead when mapping quantum algorithms from the RevLib library onto the crossbar architecture. The three axes correspond to benchmark characteristics, namely, number of gates [5 - 1400], number of qubits [3 - 16] and two-qubit gate percentage [18.75 - 100].

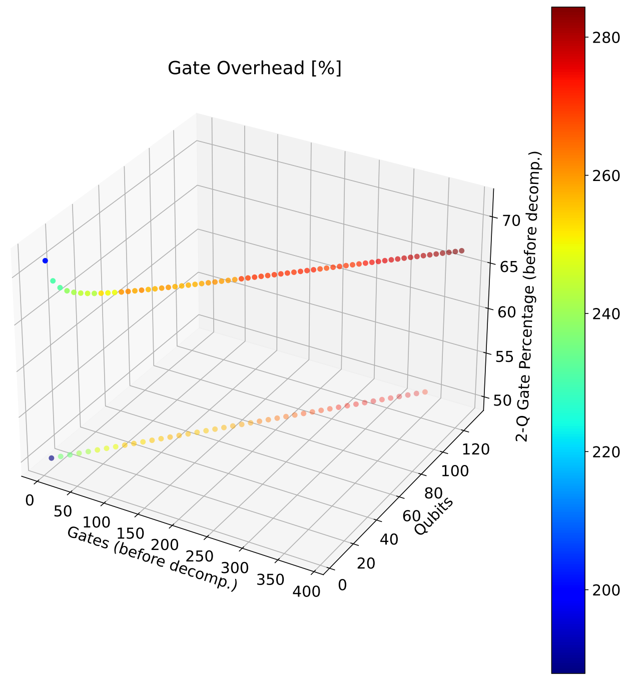


FIG. 10: Resulting gate overhead when mapping the Cuccaro Adder (top line of data points) and the Vbe Adder (bottom) quantum algorithms from the Qlib library onto the crossbar architecture. The three axes correspond to benchmark characteristics, namely, number of gates [4 - 385], number of qubits [4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

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, as it produces at least 3 additional gates compared to 4 additional gates for each shuttle-based SWAP. This is due to the unprecedented semi-global rotation scheme which is the first time that single-qubit gates require additional instructions (i.e., produce gate overhead) compared to other qubit architectures. 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, that could result in higher ESP. Additionally, such a heuristic algorithm can allow multiple control or target qubits ([10]) to be shuttled around the topology allowing for parallelization of many two-qubit gates while avoiding high error variability in the topology [18]. However, such a solution must be implemented with 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 without the need to apply any scheme afterwards (see the example in Sec. IV).

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 gates (due to the semi-global rotation scheme), an extension of an initial placement algorithm accounting for single-qubit gates can reduce the gate overhead.

Last but not least, we have emphasized that to concretely evaluate results, there has to be sufficient characterization of

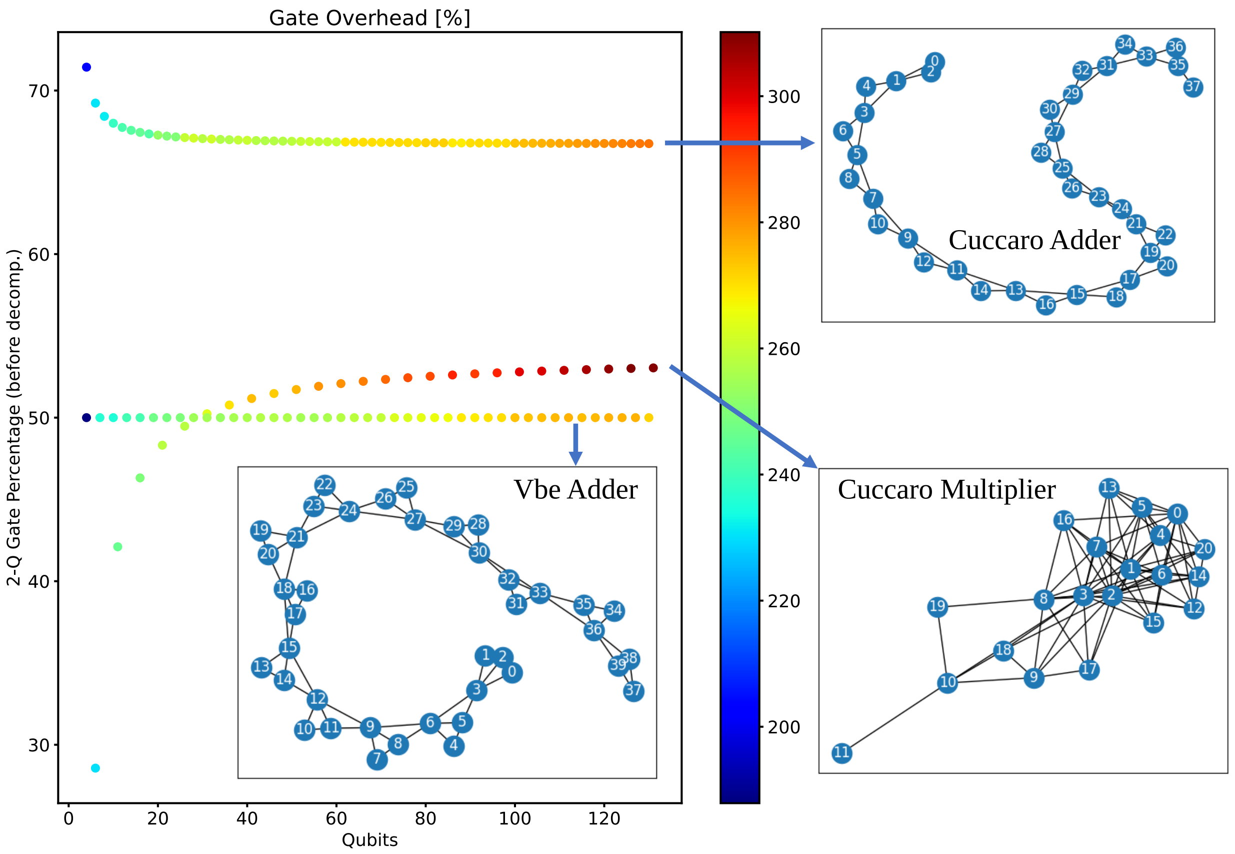


FIG. 11: Resulitng gate overhead when the Vbe Adder, Cuccaro Adder and Cuccaro Multiplier from the Qlib library are mapped onto the crossbar architecture alongside their Quantum Interaction Graphs (QIG) consisting of 40, 38 and 21 qubits, respectively. The y-axis represents the two-qubit gate percentage 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.

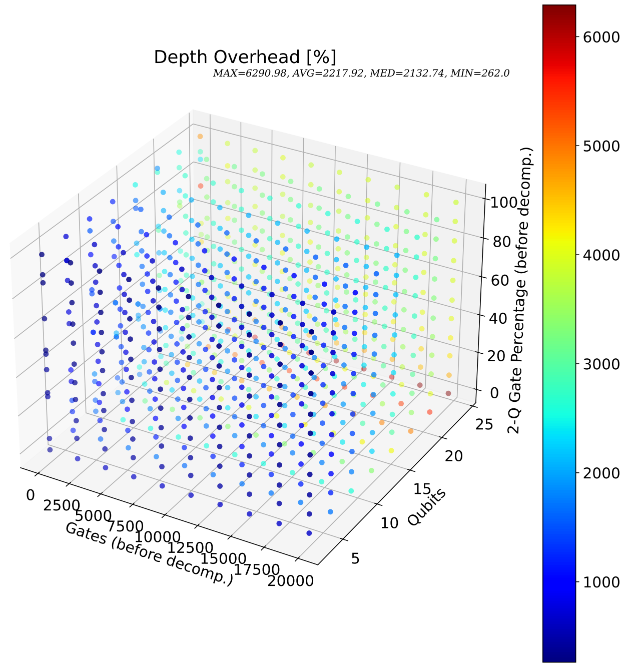
benchmarks, especially when evaluating novel architectures and mapping techniques. In our analysis, we did not rely only on simple benchmark parameters, such as the percentage of two-qubit gates, but also on the internal structure of benchmarks using the Quantum Interaction Graph (QIG).

C. Depth Overhead

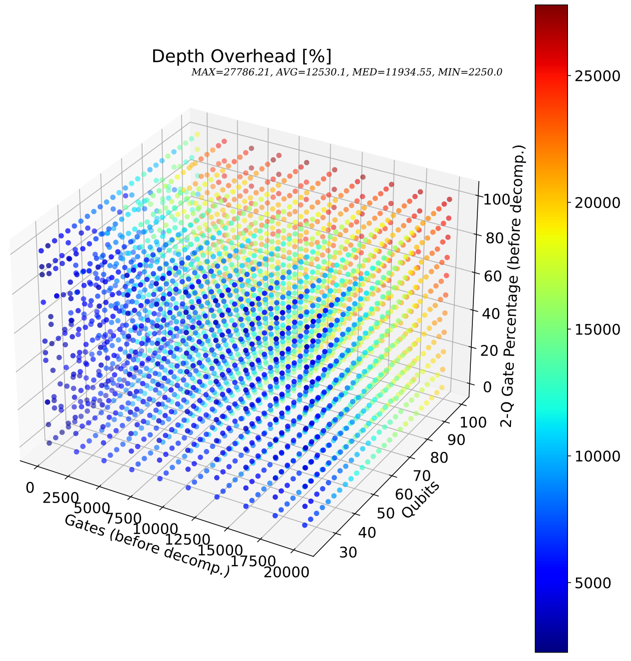
This time, we analyse the depth overhead when mapping onto the crossbar the same random uniform benchmark set as in Fig. 8. In Fig. 12, it can be observed that the trend (colours) of the depth overhead changes for different ranges of number of qubits as shown in the two subfigures. Knowing that the main source of depth overhead originates from X or Y gates (at least 3 additional cycles), we expect the depth overhead to become higher in lower regions of two-qubit gate percentage. That is observed in Fig. 12a, where the number of qubits goes up to 25. However, moving on to Fig. 12b, 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. This is apparent by its increase (from blue to red colour) 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, Fig. 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 con-



(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%].

tribute to this overhead as discussed. It is then important to have a closer look at the X or Y rotation gate percentage and further analyse 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, $S\dagger$, $Ry(\frac{-\pi}{2})$. Thus, it is relevant to consider this change in gate percentage 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 Fig. 14 we show the depth overhead of the Cuccaro Adder (upper dots) and the Vbe Adder (lower dots) with the same ranges as in Fig. 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.

D. 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 favoured scalability for Cuccaro Adder in Fig. 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 stated before, the fact that gate overhead 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 Sec. VII B. From an architectural point of view, since the shuttle operation is so relevant, there have to be as few operational

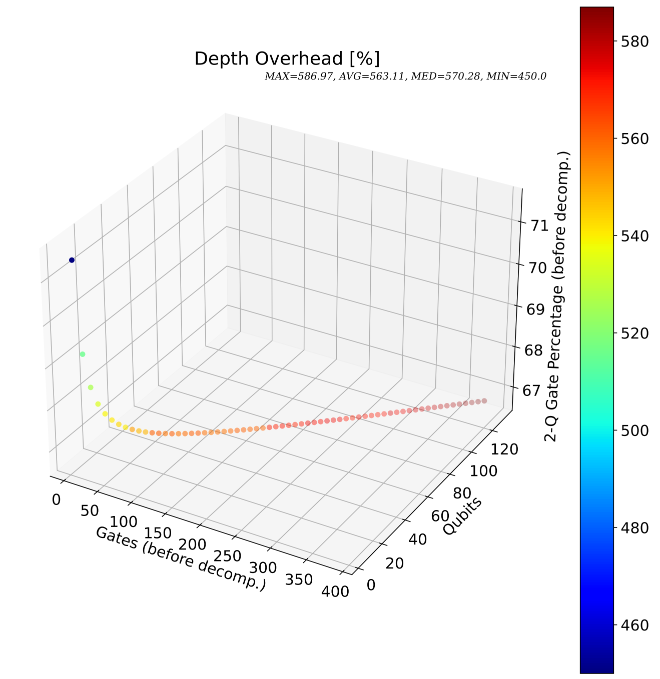


FIG. 13: Resulting depth overhead when Cuccaro Adder from the Qlib library is mapped onto the crossbar architecture. The three axes correspond to benchmark characteristics, namely, number of gates [4 - 385], number of qubits [4 - 130] and two-qubit gate percentage [66.75 - 71.43].

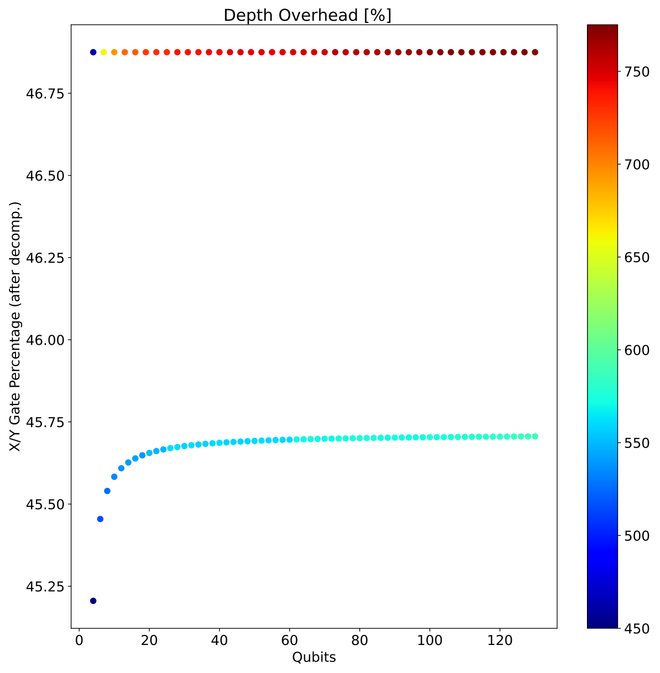


FIG. 14: Resulting depth overhead when Cuccaro Adder (bottom line of data points) and Vbe Adder (top) from the Qlib library are mapped onto the crossbar architecture. The y-axis represents the X or Y gate percentage, and the x-axis the number of qubits.

constraints as possible when mapping them.

E. 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 [1]) and 99.98% for $\sqrt{SWAP}$s, the ESP drops drastically to 0 in most algorithms with a high number of gates. For that reason, we just focused on the Bernstein-Vazirani algorithm as it has got a low percentage of two-qubit gates (usually there are only one or two $CNOT$s), therefore the error is mostly introduced by single-qubit gates.

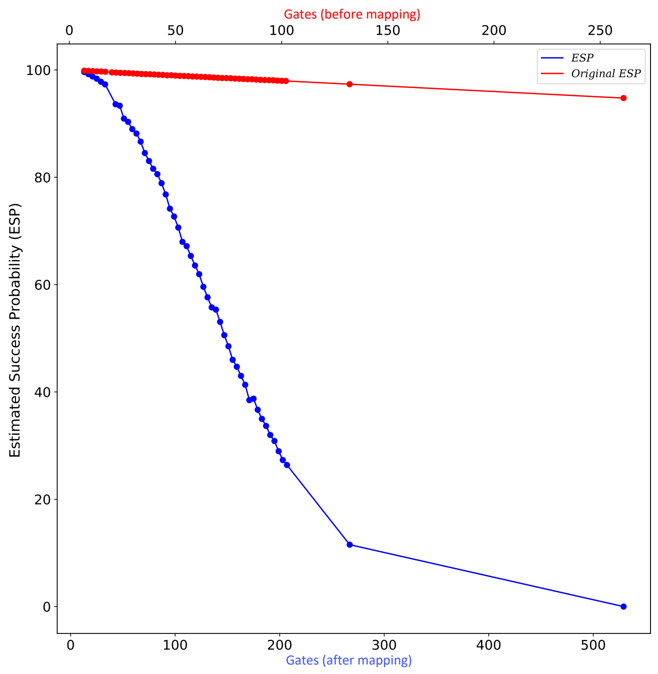


FIG. 15: Estimated success probability (ESP) before and after compilation of Bernstein-Vazirani algorithm from 2 to 129 qubits.

Fig. 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 Sec. IV 2.

F. Insights from Estimated Success Probability analysis

Our estimated success probability equation 2, although simple, is approximating a worse-case-scenario algorithm success rate. 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 [28, 44]. 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.

On real NISQ quantum devices there are other sources of noise noise that impact algorithm execution. Fortunately, it is expected that processors will gradually become more robust with better fabrication tolerances and improved error-mitigation and mapping techniques will be developed and ultimately quantum error correction protocols will be used. It remains challenging, however, to accurately simulate errors in large-scale devices to derive algorithm's success probability.

G. Compilation time

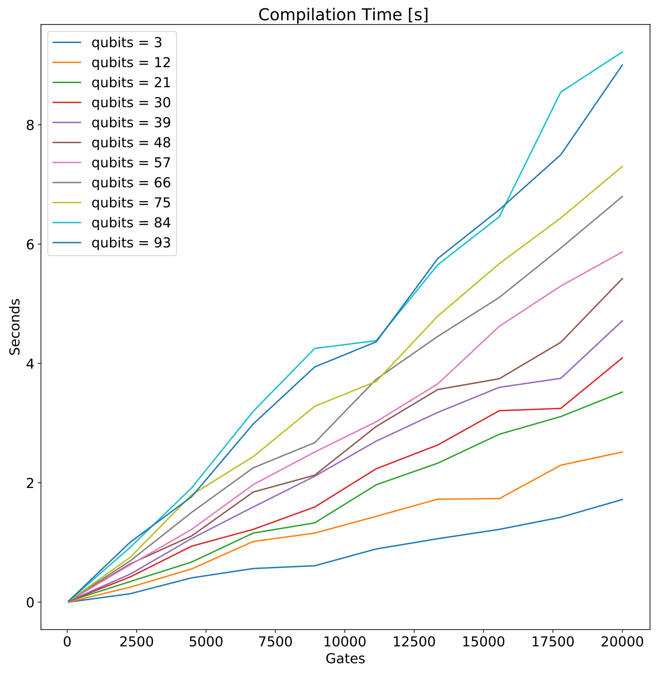


FIG. 16: Compilation time when mapping random uniform algorithms with 50% of two-qubit gates onto the crossbar architecture. We observe a linear relation which makes *SpinQ* suitable for scalable spin-qubit crossbar architectures.

Finally, we measure the compilation time of our solution to evaluate its scalability. The compilation time of *SpinQ Integrated Strategy* can be seen in Fig. 16 for a subset of the random uniform circuits that have been used in Fig. 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 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 can be directed towards reducing the slopes for each qubit count.

VIII. DISCUSSION AND FUTURE DIRECTIONS

TABLE I: Computational complexity comparison between compilation strategies for the crossbar architecture [1]. With n we denote the number of gates in a quantum circuit.

Strategy	Complexity
Backtrack [27]	$O(n^3)$
Suffer a side effect [27]	$O(n^2 log(n))$
Avoid the deadlock [27]	$O(n)$
Integrated (ours)	$O(n)$

Integrated strategy improvements. 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 large scale we have to consider the trade-off between complexity (computation time as sizes increase) and performance (less overhead and higher ESP).

Improvements in category (a) will involve a constraint and conflict check for any shuttle-based type gate to enable complete parallelization of all single-qubit gates within the second pass. Note that, 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 VII B, VII D and VII F, which can be extended to other scalable spin-qubit architectures. This will enable complete parallelization of two-qubit gates and less routing for both, one- and two-qubit gates.

Strategy Comparisons. As we discussed in Sec. IV, the crossbar architecture comes with constraints that prevent full parallelization of quantum instructions. The crossbar, however, may reach two types of conflicts (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 [27]. Table I compares the computational complexity of these three strategies with our own. The *backtrack* strategy suggested in [27] avoids conflicts by trying a different scheduling combination. If after repeating this process the scheduler has backtracked to the first instruction of the cycle (no more scheduling combinations), a new routing path is given by the routing algorithm and the scheduling is repeated. This strategy can be quite complex as the worst case scenario can un-route and un-schedule all the gates going back to a completely unmapped circuit. 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 only following the "backtracking" strategy. The final strategy, and the one implemented in [27], is called *avoid the deadlock*. This strategy, similar 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 can not 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 can not be easily improved while keeping the same complexity. Our strategy is able to maintain the same $O(n)$ complexity even after improvements.

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 maybe 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. By that time, optimizing mapping techniques for specific hardware and/or algorithm might not be as relevant as today, but rather how fast and adaptable they are to a plethora of quantum algorithms and increased number of qubits.

IX. 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 rapidly scale 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 which 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 possibly increase the performance while taking into account the compilation scalability. We also emphasized the importance of characterizing benchmarks before and after decomposition and by including their QIG structure to better evaluate results.

X. ACKNOWNLEDGEMENT

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.

[1] R. Li, L. Petit, D. P. Franke, J. P. Dehollain, J. Helsen, M. Steudtner, N. K. Thomas, Z. R. Yoscovits, K. J. Singh, S. Wehner, *et al.*, A crossbar network for silicon quantum dot qubits, Science advances **4**, eaar3960 (2018).

[2] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, *et al.*, Quantum supremacy using a programmable superconducting processor, Nature **574**, 505 (2019).

[3] L. S. Madsen, F. Laudenbach, M. F. Askarani, F. Rortais, T. Vincent, J. F. Bulmer, F. M. Miatto, L. Neuhaus, L. G. Helt, M. J. Collins, *et al.*, Quantum computational advantage with a programmable photonic processor, Nature **606**, 75 (2022).

[4] H.-Y. Huang, M. Broughton, J. Cotler, S. Chen, J. Li, M. Mohseni, H. Neven, R. Babbush, R. Kueng, J. Preskill, *et al.*, Quantum advantage in learning from experiments, Science **376**, 1182 (2022).

[5] S. Bravyi, O. Dial, J. M. Gambetta, D. Gil, and Z. Nazario, The future of quantum computing with superconducting qubits, Journal of Applied Physics **132**, 160902 (2022).

[6] J. Preskill, Quantum computing in the nisq era and beyond, Quantum **2**, 79 (2018).

[7] T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. L. O'Brien, Quantum computers, nature **464**, 45 (2010).

[8] C. G. Almudever, L. Lao, X. Fu, N. Khammassi, I. Ashraf, D. Iorga, S. Varsamopoulos, C. Eichler, A. Wallraff, L. Geck, *et al.*, The engineering challenges in quantum computing, in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017* (IEEE, 2017) pp. 836–845.

[9] A. Zulehner, A. Paler, and R. Wille, An efficient methodology for mapping quantum circuits to the ibm qx architectures, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **38**, 1226 (2018).

[10] L. Lao, H. van Someren, I. Ashraf, and C. G. Almudever, Timing and resource-aware mapping of quantum circuits to superconducting processors, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2021).

[11] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers, in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (2019) pp. 1015–1029.

[12] L. Lao and D. E. Browne, 2qan: A quantum compiler for 2-local qubit hamiltonian simulation algorithms, in *Proceedings of the 49th Annual International Symposium on Computer Architecture* (2022) pp. 351–365.

[13] S. Nishio, Y. Pan, T. Satoh, H. Amano, and R. V. Meter, Extracting success from ibm's 20-qubit machines using error-aware compilation, ACM Journal on Emerging Technologies in Computing Systems (JETC) **16**, 1 (2020).

[14] M. Bandic, S. Feld, and C. G. Almudever, Full-stack quantum computing systems in the nisq era: algorithm-driven and hardware-aware compilation techniques, in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (IEEE, 2022) pp. 1–6.

[15] P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete, Full-stack, real-system quantum computer studies: Architectural comparisons and design insights, in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)* (IEEE, 2019) pp. 527–540.

[16] N. Quetschlich, L. Burgholzer, and R. Wille, Predicting good quantum circuit compilation options, arXiv preprint arXiv:2210.08027 (2022).

[17] M. Steinberg, S. Feld, C. G. Almudever, M. Marthaler, and J.-M. Reiner, A noise-aware qubit mapping algorithm evaluated via qubit interaction-graph criteria, arXiv preprint arXiv:2103.15695 (2021).

[18] S. S. Tannu and M. K. Qureshi, Not all qubits are created equal: a case for variability-aware policies for nisq-era quantum computers, in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (2019) pp. 987–999.

[19] M. G. Pozzi, S. J. Herbert, A. Sengupta, and R. D. Mullins, Using reinforcement learning to perform qubit routing in quantum compilers, arXiv preprint arXiv:2007.15957 (2020).

[20] F. A. Zwanenburg, A. S. Dzurak, A. Morello, M. Y. Simmons, L. C. L. Hollenberg, G. Klimeck, S. Rogge, S. N. Coppersmith, and M. A. Eriksson, Silicon quantum electronics, Rev. Mod. Phys. **85**, 961 (2013).

[21] D. Loss and D. P. DiVincenzo, Quantum computation with quantum dots, Phys. Rev. A **57**, 120 (1998).

[22] L. Vandersypen, H. Bluhm, J. Clarke, A. Dzurak, R. Ishihara, A. Morello, D. Reilly, L. Schreiber, and M. Veldhorst, Interfacing spin qubits in quantum dots and donors—hot, dense, and coherent, npj Quantum Information **3**, 1 (2017).

[23] M. Veldhorst, C. Yang, J. Hwang, W. Huang, J. Dehollain, J. Muhonen, S. Simmons, A. Laucht, F. Hudson, K. M. Itoh, *et al.*, A two-qubit logic gate in silicon, Nature **526**, 410 (2015).

[24] D. Zajac, T. Hazard, X. Mi, K. Wang, and J. R. Petta, A reconfigurable gate architecture for si/sige quantum dots, Applied Physics Letters **106**, 223507 (2015).

[25] T. Watson, S. Philips, E. Kawakami, D. Ward, P. Scarlino, M. Veldhorst, D. Savage, M. Lagally, M. Friesen, S. Coppersmith, *et al.*, A programmable two-qubit quantum processor in silicon, nature **555**, 633 (2018).

[26] F. Borsoi, N. W. Hendrickx, V. John, S. Motz, F. van Riggelen, A. Sammak, S. L. de Snoo, G. Scappucci, and M. Veldhorst, Shared control of a 16 semiconductor quantum dot crossbar array, arXiv preprint arXiv:2209.06609 (2022).

[27] A. Morais Tejerina, Mapping quantum algorithms in a crossbar architecture (2019).

[28] J. Helsen, M. Steudtner, M. Veldhorst, and S. Wehner, Quantum error correction in crossbar architectures, Quantum Science and Technology **3**, 035005 (2018).

[29] C. Gidney and M. Ekerå, How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits, Quantum **5**, 433 (2021).

[30] S. Resch and U. R. Karpuzcu, Quantum computing: an overview across the system stack, arXiv preprint arXiv:1905.07240 (2019).

[31] A. Chatterjee, P. Stevenson, S. De Franceschi, A. Morello, N. P. de Leon, and F. Kuemmeth, Semiconductor qubits in practice, Nature Reviews Physics **3**, 157 (2021).

[32] D. P. Franke, J. S. Clarke, L. M. Vandersypen, and M. Veldhorst, Rent's rule and extensibility in quantum computing, Microprocessors and Microsystems **67**, 1 (2019).

[33] M. Meyer, C. Déprez, T. R. van Abswoude, D. Liu, C.-A. Wang, S. Karwal, S. Oosterhout, F. Borsoi, A. Sammak, N. W. Hendrickx, *et al.*, Electrical control of uniformity in quantum dot devices, arXiv preprint arXiv:2211.13493 (2022).

[34] J. M. Boter, J. P. Dehollain, J. P. Van Dijk, Y. Xu, T. Hensgens, R. Versluis, H. W. Naus, J. S. Clarke, M. Veldhorst, F. Sebas-

tiano, *et al.*, Physical Review Applied **18**, 024053 (2022).

[35] C. D. Hill, E. Peretz, S. J. Hile, M. G. House, M. Fuechsle, S. Rogge, M. Y. Simmons, and L. C. Hollenberg, A surface code quantum computer in silicon, Science advances **1**, e1500707 (2015).

[36] B. Paquelet Wuetz, P. Bavdaz, L. Yeoh, R. Schouten, H. Van Der Does, M. Tiggelman, D. Sabbagh, A. Sammak, C. G. Almudever, F. Sebastiano, *et al.*, Multiplexed quantum transport using commercial off-the-shelf cmos at sub-kelvin temperatures, npj Quantum Information **6**, 1 (2020).

[37] S. Pauka, K. Das, R. Kalra, A. Moini, Y. Yang, M. Trainer, A. Bousquet, C. Cantaloube, N. Dick, G. Gardner, *et al.*, A cryogenic interface for controlling many qubits, arXiv preprint arXiv:1912.01299 (2019).

[38] M. Veldhorst, H. Eenink, C.-H. Yang, and A. S. Dzurak, Silicon cmos architecture for a spin-based quantum computer, Nature communications **8**, 1 (2017).

[39] N. W. Hendrickx, W. I. Lawrie, M. Russ, F. van Riggelen, S. L. de Snoo, R. N. Schouten, A. Sammak, G. Scappucci, and M. Veldhorst, A four-qubit germanium quantum processor, Nature **591**, 580 (2021).

[40] M. Veldhorst, J. Hwang, C. Yang, A. Leenstra, B. de Ronde, J. Dehollain, J. Muhonen, F. Hudson, K. M. Itoh, A. Morello, *et al.*, An addressable quantum dot qubit with fault-tolerant control-fidelity, Nature nanotechnology **9**, 981 (2014).

[41] T. Fujita, T. A. Baart, C. Reichl, W. Wegscheider, and L. M. K. Vandersypen, Coherent shuttle of electron-spin states, npj Quantum Information **3**, 1 (2017).

[42] G. Li, Y. Ding, and Y. Xie, Tackling the qubit mapping problem for nisq-era quantum devices, in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (2019) pp. 1001–1014.

[43] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, Trapped-ion quantum computing: Progress and challenges, Applied Physics Reviews **6**, 021314 (2019).

[44] Y. Kharkov, A. Ivanova, E. Mikhantiev, and A. Kotelnikov, Arline benchmarks: Automated benchmarking platform for quantum compilers, arXiv preprint arXiv:2202.14025 (2022).

[45] A. Sinha, U. Azad, and H. Singh, Qubit routing using graph neural network aided monte carlo tree search, in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36 (2022) pp. 9935–9943.

[46] M. Bandic, H. Zarein, E. Alarcon, and C. G. Almudever, On structured design space exploration for mapping of quantum algorithms, in *2020 XXXV conference on design of circuits and integrated systems (DCIS)* (IEEE, 2020) pp. 1–6.

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

[48] D. M. A. L. Valada, Predicting the fidelity of quantum circuits search for better metrics for the qubit mapping problem (2020).

[49] IBM, Qiskit aer library, https://qiskit.org/documentation/apidoc/aer_library.html (2022).

[50] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, t— ket¿: A retargetable compiler for nisq devices, Quantum Science and Technology (2020).

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

[52] C.-C. Lin, A. Chakrabarti, and N. K. Jha, Qlib: Quantum module library, ACM Journal on Emerging Technologies in Computing Systems (JETC) **11**, 1 (2014).