Hardware-aware Compilation for Different Quantum Computing Platforms

Ludwig Schmid, Daniel Schoenberger, Yannick Stade, Lukas Burgholzer, and Robert Wille

Abstract Quantum computing is rapidly advancing toward practical utility. But current hardware platforms face significant physical constraints. These include limited qubit connectivity, noise, and architectural restrictions, necessitating sophisticated compilation strategies. This chapter presents an overview of hardware-aware quantum circuit compilation tailored to each of the three major hardware platforms: superconducting qubits, neutral atoms, and trapped ions. We discuss key compilation stages—mapping, routing, and scheduling—emphasizing how platform-specific features influence algorithm design. Methods include optimization based on satisfiability solvers and heuristic approaches like A* search. All discussed techniques are available in the open-source *Munich Quantum Toolkit* (MQT), facilitating further community development. The discussion underlines the importance of architecture-specific compilation and suggests how these approaches can extend to further platforms such as photonic and spin-based systems.

Key words: Quantum Computing, Compilation, Superconducting, Trapped Ions, Neutral Atoms, Routing, Mapping

1 Introduction

Quantum computing is transitioning from theoretical promise to practical realization, with hardware platforms becoming increasingly capable and scalable. However, these systems still operate under significant constraints, including limited connectivity,

Ludwig Schmid, Daniel Schoenberger, Yannick Stade, Lukas Burgholzer, and Robert Wille
Technical University of Munich, Chair for Design Automation, Munich, Germany
Lukas Burgholzer and Robert Wille
Munich Quantum Software Company GmbH, Garching near Munich, Germany
Robert Wille
Software Competence Center Hagenberg GmbH, Hagenberg, Austria

noise, and architectural restrictions that hinder the direct execution of arbitrary quantum circuits [80].

To bridge the gap between high-level quantum programs and the physical realities of hardware, advanced software tools are indispensable [18, 98]. Among these, *quantum circuit compilation* plays a central role: it translates abstract quantum algorithms into concrete sequences of hardware-compatible operations, while optimizing for platform-specific constraints and performance objectives. This process must be tightly coupled with the physical characteristics of the target architecture, as different platforms offer distinct native operations and constraints.

Effective compilation strategies must therefore be hardware-aware. What is optimal for one platform may be inefficient or even infeasible for another. More precisely:

(1) **Superconducting systems** present challenges due to static and sparse qubit connectivity. Compilation must account for a fixed coupling map, often requiring careful insertion of `SWAP` gates to enable necessary interactions [19, 58, 106].
(2) **Neutral atom platforms** offer more flexibility through atom rearrangement, enabling dynamic qubit layouts [88]. However, compilation must respect physical constraints like non-crossing movements and designated zones [9, 94].
(3) **Trapped-ion devices**, particularly those using the Quantum Charge Coupled Device (QCCD) architecture [45], require careful scheduling of ion transport and gate execution, balancing latency and architectural limitations [50, 90].

These variations necessitate tailored compilation pipelines that exploit each platform's strengths while mitigating its limitations.

In this chapter, we present an overview of quantum circuit compilation across these three leading hardware platforms. We focus on three key stages of the compilation process that are particularly influenced by hardware constraints:

- **Mapping** – Assigns logical qubits to physical hardware qubits, ensuring that the circuit can be executed within the device's connectivity structure.
- **Routing** – Introduces additional operations—such as `SWAP` or `MOVE` gates—to enable necessary two-qubit interactions between non-adjacent qubits.
- **Scheduling** – Organizes operations over time to respect hardware-specific execution constraints and to optimize overall performance.

Together with **Synthesis** of the gate sequence and a final **Evaluation**, these steps form the core of hardware-aware quantum compilation.

All discussed techniques are implemented in the open-source as part of the *Munich Quantum Toolkit* [109] (MQT), giving easy access to the community to use and improve the available tools.

The remainder of this chapter is structured as follows. Section 2 offers an overview of quantum circuit compilation, covering key subroutines: synthesis, mapping, routing, scheduling, and evaluation. Section 3 to Section 5 focus on hardware-aware compilation for specific quantum platforms. Section 3 addresses superconducting chips, detailing routing challenges and symbolic SWAP optimization. Section 4 discusses neutral atom platforms, highlighting compilation strategies for hybrid and zoned architectures under physical and optical constraints. Section 5 explores trapped

ion systems with a focus on time-optimal shuttling in shuttling architectures. Section 6 briefly surveys other quantum platforms and their compilation considerations. Finally, Section 7 concludes with a summary and future outlook.

2 Quantum Circuit Compilation

Broadly speaking, the term "compilation" refers to the transformation of a high-level, abstract quantum algorithm description into a lower-level representation comprising operations suitable for further processing or direct hardware execution [18, 27, 33, 64]. This process is typically realized through a layered software architecture, where each layer is dedicated to handling specific compilation tasks. Due to the interdisciplinary nature of quantum computing, which merges concepts from both computer science and physics, terminology—including the term "compilation" itself—may vary depending on the context. To mitigate any ambiguity of the term *compilation*, we provide a brief overview and clarification of the compilation subroutines considered in this chapter.

2.1 Compilation Subroutines

The compilation subroutines can be grouped into three principal categories, classified according to their level of abstraction and their dependence on a specific hardware architecture. These three categories are adapted from similar considerations for classical compilation for different hardware architectures [18].

- *Platform-independent compilation*
  This category comprises high-level optimization techniques that are applicable regardless of the target quantum platform. Examples include loop unrolling and function inlining [43], which simplify constructs such as for-loops and function invocations in a manner analogous to classical compiler optimizations. Additional techniques in this category involve gate-level optimizations, such as exploiting commutation rules to reduce gate count or depth [40, 65].
- *Platform-dependent compilation*
  These subroutines are tailored to the specific features and constraints of a given quantum hardware platform. The resulting output is a sequence of platform-specific instructions, which remain abstracted from the actual physical hardware implementation.
- *Device-dependent compilation*
  This lowest layer of compilation is customized for a concrete hardware configuration, or even a specific device. It converts platform-specific operations into hardware-level instructions executable by the quantum processing unit. It is sometimes referred to as *firmware*, emphasizing its close integration with the hardware.

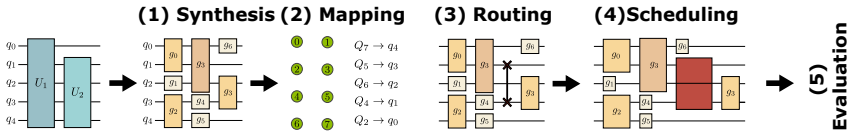


Fig. 1: The five steps of hardware-dependent compilation, adapted from [88].

This work focuses on the platform-dependent compilation subroutines, with a particular emphasis on the stages of *Mapping* and *Routing*, which will be explained in more detail below. We analyze these steps across different quantum hardware architectures—namely, superconducting qubits, neutral atoms, and trapped ions. These platforms are characterized by fundamentally distinct physical constraints and interaction topologies, which require tailored and platform-specific compilation strategies.

While this general structure above applies also to many other compilation schemes, such as compilation for measurement-based quantum computing (MBQC) and compilation for machines operating on a fault-tolerant level using quantum error correction (QEC), our focus here lies on compilation for gate-based computation on near-term intermediate-scale quantum (NISQ) devices. Nevertheless, many of the principles and hardware-specific considerations discussed in this chapter can inform and be adapted to MBQC or QEC-based compilation scenarios, where similar challenges manifest at different levels of abstraction.

From here on, we assume that both hardware-dependent and platform-independent optimization steps have been completed in advance. This can be achieved using e.g. ZX-based tools such as `PyZX` [46]. Our aim now is to produce hardware-oriented instruction sequences while avoiding lower-level aspects such as signal control or pulse-level implementation, which depend to an even higher degree on the specific hardware setup. This setting naturally leads to five core tasks within the compilation chain, illustrated in Figure 1.

(1) *Synthesis*, also commonly termed *decomposition*, concerns the breakdown of abstract quantum gates into operations that are natively supported by the target platform.
(2) *Mapping* addresses the spatial allocation of circuit qubits onto physical hardware qubits.
(3) *Routing* ensures the necessary qubit connectivity is achieved by inserting connectivity-altering operations such as `SWAP` or `MOVE` operations. This enables two-qubit interactions across initially unconnected physical qubits.
(4) *Scheduling* involves the temporal organization of operations, ensuring that the platform's architectural constraints are fulfilled while achieving an optimized execution of the operations.
(5) *Evaluation* assesses the quality of the compilation output based on appropriate figures of merit. These metrics may vary depending on the abstraction level and the target hardware platform, and may include measures such as circuit depth, gate count, estimated fidelity, execution time, or resource overhead.

The subsequent sections provide more formal definitions and some examples of these five steps, as applicable to the context of this chapter.

2.1.1 Synthesis

In the *quantum circuit model*, any non-dissipative quantum computation $U \in \mathbb{C}^{2^n \times 2^n}$ can be represented as a finite sequence of unitary operations $g \in \mathbb{C}^{2^m \times 2^m}$, with $m < n$. These smaller unitary operations are known as *quantum gates*. Each gate acts on a subset of qubits, with n (m) denoting the number of qubits involved in the overall circuit (individual gate), respectively. These gates are restricted to reversible unitary transformations and therefore are closely related to classical reversible gates and the corresponding logic, such that many of the classical synthesis methods can be directly mapped to the quantum case [78].

But while in the classical world one only has to consider operations on the binary computational states, quantum algorithms exhibit arbitrary unitary rotations around the Bloch sphere, requiring some kind of discretization. The Solovay-Kitaev theorem [47] ensures that for a general unitary operation U, an approximation up to an arbitrarily small error can always be obtained efficiently using only a discrete set of finite size. Such a collection of gates is referred to as an *universal gate set* Σ^{univ}. On real hardware, however, only a limited set of operations can be implemented directly, which is specified by the *native gate set* Σ^{native}. This native set is typically chosen to perform operations that closely represent the physical behavior of the system. To have a *universal quantum computer*, one requires Σ^{nativ} to also be a universal gate set.

Typically, this is realized by Σ^{native} comprising single-qubit rotations around at least two different axes and at least one entangling two-qubit gate, such as CX, CZ, or the Mølmer–Sørensen gate [15]. To then enable execution on a particular quantum device, any quantum operation not included in the native gate set must be decomposed into a sequence the native gates, a process commonly referred to as *synthesis* or *decomposition* [1, 28, 66, 67].

Definition 1 (Synthesis) Given a unitary quantum operation $U \in \mathbb{C}^{2^n \times 2^n}$ and a native gate set Σ^{native}, the *synthesis* task consists in finding a gate sequence

$$\tilde{U} = g_{N-1} \circ \cdots \circ g_0$$

with all $g_0, \ldots, g_{N-1} \in \Sigma^{\text{native}}$ such that $\tilde{U}$ approximates U up to a specified precision.

Example 1 Examples of native gate sets for selected quantum platforms, depending mostly on the way the gates are realized on hardware:

- IBM (until 2021): $[U_3(\theta, \phi, \lambda), CX]$,
- IBM (until 2024): $[Id, R_Z(\theta), \sqrt{X}, X, CX]$,
- IBM (current): $[Id, R_Z(\theta), \sqrt{X}, X, CZ]$,
- IonQ/AQT: $[R_X(\theta), R_Y(\theta), R_Z(\theta), R_{XX}(\theta)]$.

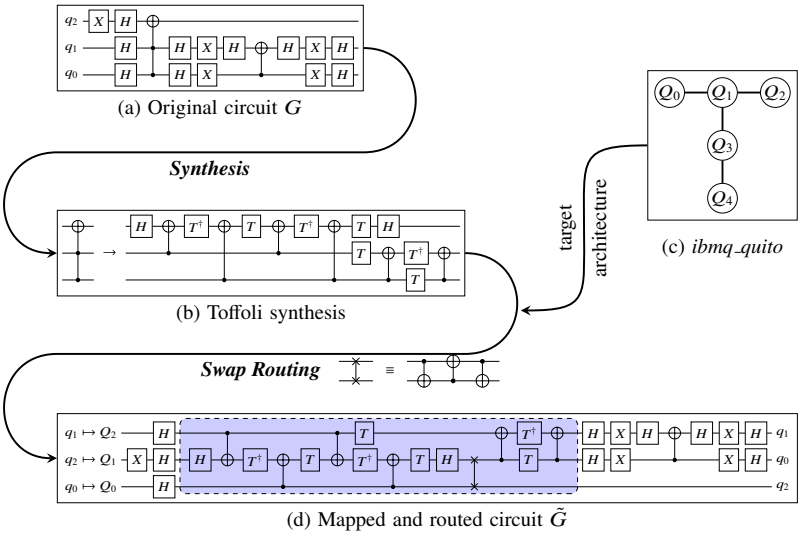


Fig. 2: Exemplary compilation-flow illustration for a Grover-type circuit G.

As an example, none of these gate sets support operations acting on more than two qubits directly. Hence, all larger multi-qubit operations, including three-qubit gates such as the Toffoli gate, must be decomposed into single- and two-qubit gates.

Example 2 Consider the circuit depicted in Figure 2a, which implements a simplified version of Grover's search algorithm. This circuit contains a three-qubit Toffoli gate, which cannot be implemented natively on any of the above platforms. Consequently, synthesis must be applied to decompose the Toffoli gate into native operations. Figure 2b (b) illustrates one such decomposition using the Clifford+T gate set.

2.1.2 Mapping

After synthesis, the next step in the compilation flow is *mapping*, which establishes a correspondence between the logical circuit qubits $\mathbf{Q} = \{q_0, \ldots, q_{n-1}\}$ and the physical hardware qubits $\mathbf{P} = \{Q_0, \ldots, Q_{|\mathbf{P}|}\}$ with $|\mathbf{P}| \geq n$. This relationship is formalized as a bijective mapping function $f : \mathbf{Q} \to \mathbf{P}$.

Some *dynamic quantum processors*, such as neutral atoms or trapped ions, allow for rearrangement of the physical qubits. Here, the physical qubits $\mathbf{P}$ are not fixed in space but must be placed on a coordinate system $\mathbf{C} = \{C_\alpha\}_{\alpha=0}^{|\mathbf{C}|}$, where $|\mathbf{C}| > |\mathbf{P}| \geq n$ represent sites where physical qubits can be placed.

A good assignment must account for the constraints imposed by the hardware's physical connectivity. The connectivity is typically encoded in the form of a *coupling graph* $G = (\mathbf{P}, \mathbf{E})$, where nodes correspond to physical qubits, and edges $\mathbf{E}$ represent pairs that can directly interact through native two-qubit gates. The goal is to place

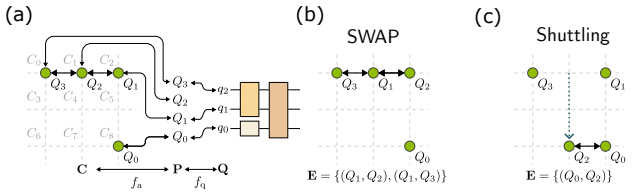


Fig. 3: Illustration [adapted from [89]] of the qubit and coordinate mapping (a) and the two routing strategies, swapping (b) and shuttling (c).

circuit qubits that frequently interact onto adjacent physical qubits in the coupling graph.

Example 3 The coupling graph of a 4-qubit superconducting chip is shown in Figure 2c.

Definition 2 (Mapping)

Given a quantum circuit $U = g_{N-1} \circ \cdots \circ g_0$ and a coupling graph $G = (\mathbf{P}, \mathbf{E})$, the task of *mapping* is to compute two bijective functions:

- **Qubit mapping** $f_\mathrm{q} : \mathbf{Q} \to \mathbf{P}$ assigning circuit qubits to physical atoms.
- **Atom mapping** $f_\mathrm{a} : \mathbf{P} \to \mathbf{C}$ placing physical qubits at coordinates.

such that as many two-qubit gates $g(q_i, q_j)$ as possible satisfy $(f_\mathrm{a} \circ f_\mathrm{q}(q_i), f_\mathrm{c} \circ f_\mathrm{q}(q_j)) \in \mathbf{E}$.

Example 4 Figure 3(a) illustrates how $n = 3$ circuit qubits are mapped to $|\mathbf{Q}| = 4$ physical qubits which are layed out on a 3×3 grid of $|\mathbf{C}| = 9$ coordinates.

However, due to the limited connectivity of most quantum hardware and the evolving interaction patterns of quantum circuits, a single static assignment rarely suffices to enable all required interactions, making dynamic adjustments of f_q or f_a necessary.

2.1.3 Routing

Because a static mapping typically does not fulfill all interactions, especially in circuits with complex entanglement structures, a dynamic adaptation step called *routing* is necessary.

Routing introduces auxiliary operations—`SWAP` and `MOVE`—to modify the mapping functions f_q and f_a from above throughout circuit execution. Inserting these operations throughout the circuit execution enables qubit interactions that are otherwise disallowed by the initial hardware topology.

A $\texttt{SWAP}(Q_i, Q_j)$ operation exchanges the logical assignments of two physical qubits, effectively updating f_q at runtime. This is implemented on the circuit level via a sequence of three `CX` gates, introducing additional circuit depth and noise.

Example 5 Figure 3 (a) shows an initial mapping of the circuit. (b) shows the mapping after applying a `SWAP` between qubit Q_1 and Q_2 changing the coupling graph edges $\mathbf{E}$.

Dynamic systems offer an alternative: `MOVE`(Q_i), which physically repositions an atom from one coordinate to another, altering the connectivity graph G without the use of entangling gates but by directly acting on f_a. While this avoids increased gate error, it can incur errors from the rearrangement operation, such as additional latency.

Example 6 In (c) of Figure 3, the action of `MOVE`Q_2 to position C_7 is illustrated, connecting now Q_2 with Q_0.

Definition 3 (Routing)

Given a quantum circuit $U = g_{N-1} \circ \cdots \circ g_0$, initial mappings f_q^0 and f_c^0, with the corresponding coupling graph $G = (\mathbf{P}, \mathbf{E})$, *routing* is the task of inserting `SWAP` and/or `MOVE` operations such that each gate $g(q_i, q_j)$ satisfies $(f_\text{c} \circ f_\text{q}(q_i), f_\text{c} \circ f_\text{q}(q_j)) \in \mathbf{E}$ at the time of execution.

Example 7 After mapping and routing the final circuit $\tilde{G}$ of Figure 2d fulfills the connectivity constraints of the architecture in Figure 2c.

2.1.4 Scheduling

The scheduling of quantum gates refers to the temporal arrangement of gate executions after the routing step. A common constraint is that each physical qubit can typically participate in only one gate per timestep.

Further limitations arise e.g. from the classical control infrastructure. In many quantum platforms, not all qubits can be independently addressed at the same time due to limitations in control electronics, such as shared signal lines or bandwidth constraints. Physical effects can also constrain parallelism. For instance, in neutral atom systems, the Rydberg blockade effect prevents nearby atoms from being simultaneously excited, effectively limiting parallel gate execution based on spatial locality [35, 85, 88].

Definition 4 (Scheduling)

Given a routed quantum circuit $U = g_{N-1} \circ \cdots \circ g_0$, *scheduling* is the task of assigning a discrete or continuous timestep t_i to each gate g_i such that:

- no qubit is involved in more than one gate at any timestep,
- hardware control constraints are satisfied,
- physical interaction constraints (e.g., Rydberg blockade) are obeyed.

For this task, gates can be considered as blocks of different sizes and widths, as illustrated in Figure 1(4). Typically minimizing circuit depth and time, this results in a Tetris-like arrangement of gates [59].

Example 8 The last step of Figure 1 illustrates the temporal arrangement of different gates by representing them as differently sized blocks.

2.1.5 Evaluation

Evaluation assesses the quality of the compilation output through *metrics*, or *figures of merit* tailored to the specific constraints and capabilities of the target hardware. Typically, metrics such as gate count are used as indicators for the routing overhead, particularly for hardware with static connectivity, such as superconducting architectures. However, in the context of dynamic architectures, including systems capable of physically reconfiguring qubit connectivity via `SWAP` or `MOVE` operations, gate count alone is insufficient to capture performance trade-offs.

To better reflect the physical realities of such systems, we adopt a more comprehensive figure of merit: the *approximate success probability P* [88]. This metric incorporates in its definition both temporal and operational aspects of the circuit execution:

$$P = \exp\left(-\frac{t_{\text{idle}}}{T_{\text{eff}}}\right) \prod_O \mathcal{F}_O\,, \quad T_{\text{eff}} = \frac{T_1 T_2}{T_1 + T_2}\,, \tag{1}$$

where $\mathcal{F}_O \in [0, 1]$ denotes the fidelity of operation O, and T_1, T_2 are the characteristic relaxation and dephasing times of the system. The *effective coherence time* T_{eff} captures the decoherence time scale. The *idle time* t_{idle} accounts for the cumulative time during which qubits are not operated on and, therefore, suffer from decoherence. Intuitively, this metric incorporates two principal considerations:

- **Decoherence errors:** For idle qubits, amplitude damping and dephasing lead to idle errors. These are captured through T_1, T_2, and summarized in T_{eff}.
- **Gate infidelities:** Executing gates, particularly multi-qubit gates, exhibit errors characterized by an average gate fidelity $\mathcal{F}_O$.

Due to its simplicity, the approximate success probability $P(U)$ provides a compact, computationally tractable, and physically informed figure of merit that allows comparing circuits at various abstraction levels and with varying hardware capabilities. Its primary advantage lies in integrating scheduling and routing consequences directly into the metric, thereby enabling more informed and platform-specific optimization strategies.

In the following, these steps are discussed for different hardware platforms, including superconducting chips, neutral atoms, and trapped ions. The focus lies on the mapping and routing tasks, which are illustrated for the different hardware constraints, followed by a short summary of possible algorithmic solutions to these problems.

3 Superconducting Chips

Superconducting chips represent a leading hardware platform for constructing quantum processors [13, 37, 49]. Unlike trapped ions or neutral atoms that operate at microscopic scales, superconducting qubits are macroscopic circuits fabricated using

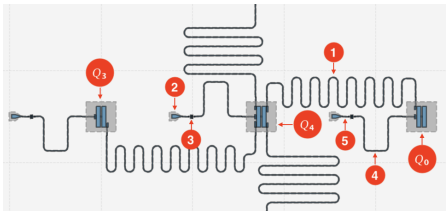


Fig. 4: Chip design with different chip components, adapted from [51].

superconducting materials. These circuits exploit the quantum mechanical properties of superconducting currents and Josephson junctions to define and control the qubits.

3.1 Background

The macroscopic design allows for detailed engineering of qubit characteristics such as energy levels, coupling strengths, and coherence times via circuit parameters [37, 82, 92, 105]. One widely used superconducting qubit design is the transmon, which minimizes charge noise and allows for relatively long coherence times. Transmons are typically implemented using a nonlinear LC oscillator formed by a Josephson junction shunted with a large capacitor, effectively reducing sensitivity to charge fluctuations[92]. The qubits are operated at millikelvin temperatures inside dilution refrigerators to maintain superconductivity and suppress thermal noise.

Qubit control and readout are achieved via microwave pulses. Single-qubit gates are performed by applying resonant microwave fields that drive Rabi oscillations between the qubit's ground and excited states. Two-qubit gates typically rely on fixed or tunable couplings mediated by microwave resonators or direct capacitive/inductive interactions. The most common gate mechanisms include cross-resonance, iSWAP, and CZ interactions, with platforms often choosing one based on gate fidelity, crosstalk, and implementation complexity [37].

In a typical architecture, qubits are connected through microwave resonators or direct couplings to enable two-qubit interactions. The specific arrangement of the coupling graph is dictated by architectural choices aiming to balance connectivity, frequency crowding, and cross-talk mitigation. Examples of common layouts are 2D topologies like square grids, IBM's heavy-hex, or Rigetti's octagonal lattice.

Example 9 Figure 4 shows a chip section created using the open-source tool Qiskit Metal [68]. It shows three trasmon qubits Q_3, Q_4, Q_0, connected by resonators ①. Each qubit also has a readout/control ② with a resonator ③ merged using a connection between capacitors and readout/control⑤.

The available gate set on superconducting platforms is restricted to a native gate set native gate set Σ^{native} as discussed in Section 2.1.1. For instance, IBM devices

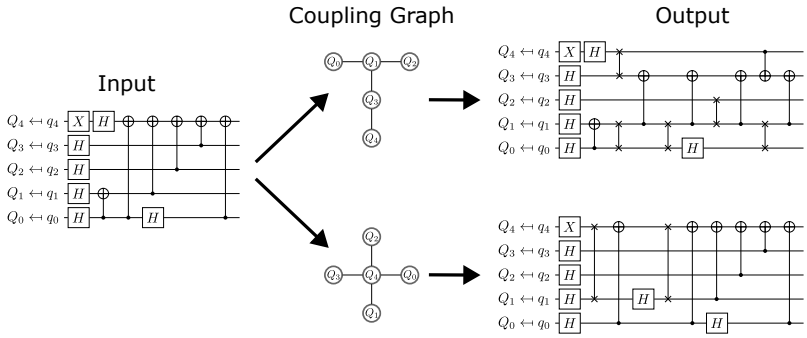


Fig. 5: Output of minimal SWAP gate insertion for two different coupling graph architectures.

support single-qubit gates such as $R_Z(\theta)$, $\sqrt{X}$, X, and two-qubit gates like CNOT, CZ.

Importantly, the connectivity of a fabricated chip is fixed, and as a result, two-qubit gates can only be directly applied between physically connected qubits. If a gate in the quantum circuit requires interaction between unconnected qubits, additional routing operations are necessary. This gives rise to the original `SWAP`-insertion problem of quantum circuit compilation.

3.2 Problem Formulation and Motivation

Superconducting quantum hardware imposes severe restrictions on qubit connectivity and allows two-qubit interactions (e.g., CNOT or CZ gates) only between specific pairs of physical qubits defined by a coupling graph. Consequently, executing arbitrary quantum circuits often requires the insertion of *SWAP gates*, which permute the positions of circuit qubits on physical hardware to enable necessary interactions.

However, each `SWAP` operation introduces a substantial overhead. In most gate libraries, a `SWAP` must be decomposed into three native two-qubit gates (e.g., CNOTs), increasing circuit depth and amplifying the impact of decoherence and gate errors. This problem of inserting the minimal number of `SWAP` gates during compilation, while respecting hardware connectivity constraints, is known to be *NP-complete* [11].

Example 10 Consider the example from Figure 5. The initial circuit to the left is mapped and routed for two different coupling graphs corresponding to two differently designed chips. The output contains the minimal number of `SWAP` gates, which can differ significantly depending on the input circuit and the coupling graph.

Existing methods typically assume a static hardware architecture, focusing on optimizing the logical-to-physical qubit mapping and gate scheduling [53, 57, 76, 99, 100, 114]. In the following, we illustrate how this can be done using satisfiability

solvers and show how to explore a broader design space that not only seeks optimal mappings for a fixed architecture but also considers potential modifications to the hardware layout.

3.3 Solution

To address the SWAP insertion problem optimally, it was proposed to use a formal and symbolic approach that fully captures the mapping constraints of a quantum circuit to a given superconducting architecture [107]. Our methodology consists of two main components: (1) an *exhaustive symbolic formulation* of the mapping problem, and (2) the *use of powerful reasoning engines*, such as SAT or SMT solvers, to efficiently navigate the resulting search space.

Given an n-qubit quantum circuit $G = g_0, g_1, \ldots, g_N$ composed of two-qubit gates, and a device described by a coupling graph $G = (\mathbf{P}, \mathbf{E})$, the goal is to find a mapping from logical to physical qubits that enables the execution of each gate as described in Definition 2 and Definition 3. We focus on minimizing the number of additional gates introduced, particularly those arising from SWAP insertions.

3.3.1 Symbolic Formulation

The mapping of logical to physical qubits is captured through *Boolean variables*. Specifically, for each gate g_k in the circuit, a variable x_{ij}^k indicates whether circuit qubit q_j is mapped to physical qubit Q_i before gate g_k. This approach accounts for the possibility of remapping (via SWAPs) at any point in the circuit.

To ensure only valid mappings are produced, we add constraints to enforce:

(1) **One-to-one mapping:** Each circuit qubit must be mapped to exactly one physical qubit, and no two circuit qubits may occupy the same physical qubit at any given time. This corresponds to the scheduling constraint from Definition 4.
(2) **Connectivity constraints:** Each two-qubit gate $g_k(q_c, q_t)$ must operate on qubits mapped to connected physical qubits in $\mathbf{E}$.

In addition, we define *permutation variables* y_π^k to model the application of SWAP operations that change the mapping between consecutive gates. These variables represent permutations $\pi \in \Pi$ of the physical qubits and are used to track when and how the mapping changes throughout the circuit.

The overall cost of a mapping, in terms of SWAP overhead, is expressed as:

$$\mathcal{F} = \sum_{k=1}^{N} \sum_{\pi \in \Pi} \text{swaps}(\pi) \cdot y_\pi^k$$

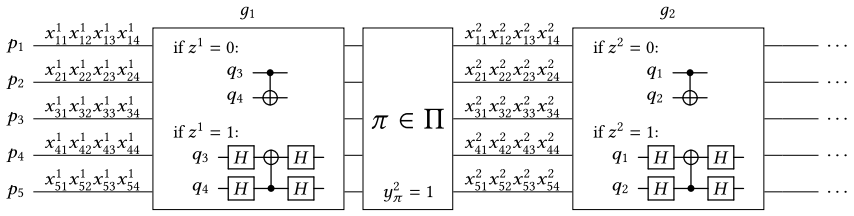


Fig. 6: Illustration of the symbolic encoding of the mapping and routing problem into binary variables, taken from [107].

where swaps(π) denotes the number of SWAPs needed to realize permutation π. The symbolic formulation can be passed to a solver with an optimization objective of minimizing $\mathcal{F}$.

Example 11 Figure 6 illustrates such an encoding for a sequence of three CX gates. The z variables are only required if the architecture is directed, as discussed below.

3.3.2 Performance Considerations

While this approach is complete and optimal, the associated computational complexity remains exponential in the worst case. To alleviate this, several performance optimizations are proposed:

- **Search space reduction** by pruning permutations that do not impact upcoming gates.
- **Constraint tightening** to eliminate invalid or redundant mappings early.
- **Omitting intermediate variables** like y^k_π in the optimization process and deriving them post hoc from x^k_{ij} assignments.

For details on these optimizations, we refer to the literature [16, 99, 107]. In the following, we want to briefly discuss some additional considerations connected to this SWAP insertion problem. All these improvements substantially lower the runtime of the symbolic mapping procedure, making it feasible to apply the method to circuits and architectures of moderate size, while still achieving near-optimal solutions [16, 107].

3.3.2.1 Considering Sub-Architectures.

When the number of circuit qubits n is less than the number of hardware qubits m, the mapping may be restricted to a subset of n physical qubits. This would require solving $\binom{m}{n}$ subproblems, each corresponding to a different n-qubit sub-architecture. To avoid this additional overhead, higher efficiency is achieved by (among other optimizations) only considering connected and non-isomorphic subarchitectures [77].

3.3.2.2 Layering Strategies.

Instead of allowing permutations before every gate, mappings may only change before selected subsets $G' \subseteq G \setminus \{g_0\}$. This significantly reduces the complexity to $2^{nm(N)}$. Several strategies for selecting G' include:

- *Disjoint qubits*: grouping gates that act on disjoint qubit sets into layers [42, 114].
- *Odd gates*: only allowing permutations before gates with odd indices.
- *Qubit triangle*: clustering gates acting on up to three qubits to exploit triangle structures in the device.

Depending on the selected G' this can significantly increase the computation at the cost of sacrificing optimality guarantees.

3.3.2.3 Improved Mappings for Undirected Architectures.

For devices with unidirectional coupling constraints, additional freedom is obtained by allowing direction-reversing transformations (e.g., replacing a reversed CNOT by surrounding it with Hadamard gates). The symbolic model introduces Boolean variables z^k to track direction reversals and extends the cost function to account for both `SWAP` and reversal operations.

3.4 Discussion

The compilation of quantum circuits for devices based on superconducting chips with a static coupling map is predominantly challenged by routing constraints, namely the insertion of `SWAP` gates due to limited qubit connectivity. These `SWAP` operations introduce significant depth and error overhead, making their minimization essential.

This section presented a symbolic and optimal approach to tackling the `SWAP` insertion problem. By encoding the mapping problem as a Boolean function and utilizing powerful reasoning engines, one can obtain provably optimal solutions [107]. To mitigate the high computational cost associated with this approach, a range of performance improvements can be employed, including sub-architecture selection, layering strategies, and direction-aware mappings [16].

It is important to note that this symbolic method is one of many strategies in the compilation landscape. Numerous heuristic approaches—such as those based on greedy algorithms, A*-search, reinforcement learning, or stochastic optimization [61, 76, 114] provide faster, albeit approximate, alternatives. These are especially relevant as the number of qubits in current quantum architectures has surpassed the size trackable by optimal approaches.

This section focused on superconducting hardware as a representative and widely used platform. The next sections will extend this investigation to other leading hardware platforms, namely neutral atom and trapped ion devices.

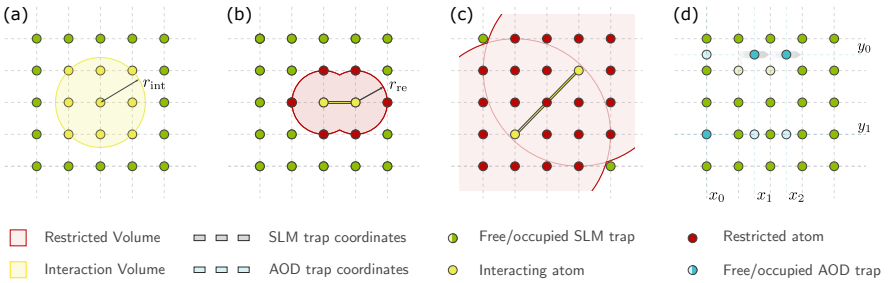


Fig. 7: Illustration of Rydberg interaction and atom rearrangements.

4 Neutral Atom Quantum Computing

Neutral atom quantum computing (NAQC) [35, 54, 87, 88, 110] utilizes individual atoms as qubits, stored in programmable arrays of optical traps. These atoms can be manipulated and entangled using lasers, allowing for scalable quantum computation. Due to their long coherence times, high-fidelity gate operations, and the ability to dynamically rearrange atomic positions, neutral atoms are a promising platform for quantum information processing.

4.1 Background

In NAQC, qubits are realized using individual atoms held in optical dipole traps, such as optical lattices or optical tweezer arrays. These traps are generated by tightly focused laser beams, creating localized potential wells in which atoms can be confined and cooled to their motional ground states.

Atom trapping is typically realized using either static optical lattices or dynamically configurable tweezer traps generated via Spatial Light Modulators (SLMs). These allow for initial stochastic loading followed by defect-free rearrangements [5, 17, 31, 32, 74].

Example 12 Figure 7 shows such optical tweezer arrays arranged in a square grid using Spatial Light Modulators (SLM). While other geometries are possible, for computational tasks, regular patterns are typically advantageous.

Qubit encoding relies on long-lived internal atomic states. In alkali atoms like Rb or Cs, the qubit states $|0\rangle$ and $|1\rangle$ are commonly represented by hyperfine ground states [30, 55]. Alkaline-earth(-like) atoms such as Sr and Yb allow for alternative encodings using metastable states, nuclear spins [4, 44, 63], or fine-structure levels [81, 102].

Single-qubit gates are implemented via laser-driven transitions between qubit states. These transitions can be induced globally across the entire array or targeted

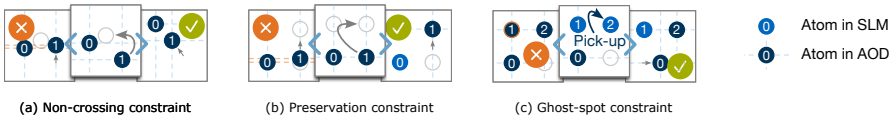


Fig. 8: Constraints on the atom movements using Acousto-Optic Deflectors (AODs).

locally to individual atoms using focused beams [9, 87]. This allows for high-fidelity state preparation and arbitrary single-qubit rotations.

Two-qubit gates leverage the Rydberg blockade mechanism [30, 86], which involves transient excitation of atoms to high-lying Rydberg states that exhibit strong dipole-dipole or van der Waals interactions [26, 70]. When one atom is excited to a Rydberg state, it shifts the energy levels of nearby atoms, preventing their simultaneous excitation. This conditional dynamic is exploited to implement entangling gates such as the controlled-Z (CZ) gate [25, 29, 39, 41, 56].

A necessary condition for this mechanism to work is that the involved qubits lie within an *interaction radius* r_{int}. Additionally, to mitigate crosstalk during parallel operations, distinct gates must be separated by at least a *restriction radius* $r_{\text{restr}} \geq r_{\text{int}}$.

Example 13 The yellow region in Figure 7 (a) indicates the possible interaction candidates for two-qubit gates for a certain interaction radius r_{int}. Figure 7 (b) shows a possible choice and the corresponding restriction radius r_{restr}. The red region corresponds to the restricted volume for other multi-qubit gates to be executed simultaneously. This restricted volume increases for long-range interactions as illustrated in Figure 7 (c).

Atom shuttling is realized by loading atoms from static SLM traps to traps created using two Acousto-Optic Deflectors (AODs) in horizontal and vertical directions. These AOD traps can be described using row and column indices with trap sites at the intersections [88, 101]. By modulating the trapping lasers, one can move rows/-columns, resulting in mobile trap sites which can be *rearranged* or *shuttled*. Several qubits can be moved simultaneously by moving multiple row/column intersections. However, three constraints, illustrated in Figure 8 must be respected [88, 94, 101]:

(a) *Non-crossing constraint:* Row and column movements must preserve ordering, i.e., they cannot cross.
(b) *Preservation constraint:* Atoms trapped in the same AOD row/column must remain in the same row/column.
(c) *Ghost-spot constraint:* Inactive AOD sites still form traps, creating *ghost spots* that must not intersect with existing qubit paths.

These are mitigated through careful loading sequences and offset maneuvers that isolate ghost spots from occupied paths.

Example 14 Figure 9 illustrates a scenario where q_0 needs to be shuttled to q_1 and q_3, q_4 next to q_2. Due to the above three constraints, this needs to be done in multiple loading and movement steps. As a first step (1), q_0 is loaded by activating

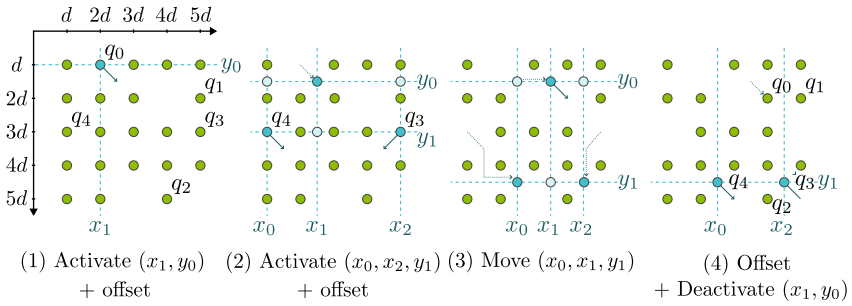


Fig. 9: Loading and unloading sequences due to AOD constraints, taken from [89].

AOD coordinates at $x_1 = 2d$ and $y_0 = d$. To prevent problematic ghost spots with the following activations, a small offset move is applied. Then (2), q_3 and q_4 can be loaded simultaneously in the same row by activating $x_0 = d$, $x_2 = 5d$ and row $y_1 = 3d$. Note that due to the previous offset, all resulting ghost spots (light blue) are in the empty inter-qubit regions. The qubits can then be moved (3) to their destinations without crossing any other AOD coordinate and placed sequentially using offset movements (4).

4.2 Problem Motivation & Description

Compilation for NAQC systems introduces novel challenges due to the physical and architectural flexibility of neutral atom platforms. Unlike static architectures such as the previous superconducting chips with fixed connectivity, NAQC requires a dynamic determination of qubit placement and routing paths, particularly under the above AOD-movement constraints. This section explores this compilation problem for two different neutral atom-based architectures: hybrid and zoned.

4.2.1 Hybrid Architectures

In *hybrid architectures*, atoms are initially arranged in static optical traps formed by a Spatial Light Modulator (SLM) and are loaded into dynamic Acousto-Optic Deflector (AOD) traps only when necessary [87, 89]. Furthermore, all atoms are assumed to be individually addressable. Two-qubit gate execution depends on the inter-atomic distance: two atoms only interact and execute the two-qubit gate if they are within the interaction radius r_{int} as discussed in Section 4.1.

We want to focus on the mapping and routing task, which is a trade-off between two routing possibilities:

- *Gate-based routing*, where connectivity is achieved by inserting `SWAP` operations to move circuit qubits closer together (similar to superconducting hardware).

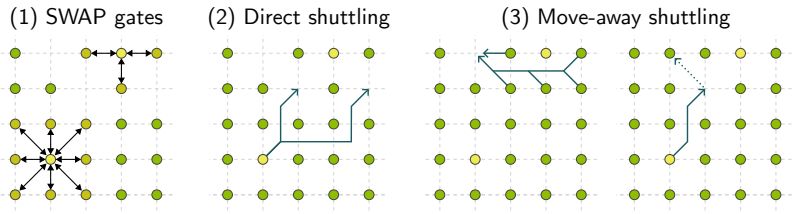


Fig. 10: Considered searchspace for routing on hybrid architectures, taken from [89].

- *Shuttling-based routing*, where atoms are physically moved to new coordinates using AOD-based traps.

In particular, considering Definition 2, gate-based mapping acts on the circuit qubit to hardware qubit mapping f_{q}, while shuttling-based routing acts on the mapping f_{a} from hardware qubits to positional coordinates.

This creates a two-fold mapping challenge with an expanded search space and the risk of mapping conflicts. Some recent works have focused exclusively on one method [2, 59, 75, 101], but a hybrid approach is necessary to fully exploit NAQC capabilities.

Example 15 Figure 10 illustrates the considered search space to connect two qubits. Shown are the possible operations for the three cases: (1) gate-based `SWAP`s, (2) directly shuttling to one of the free coordinates, and (3) requiring a move-away operation beforehand.

The individual addressability of hybrid architectures imposes a challenge on the control and focus of the Rydberg laser beam. To circumvent this, architectures with global control over a subregion of the atom registers were proposed as discussed in the following.

4.2.2 Zoned Architectures

Figure 11 depicts a schematic of a *zoned neutral atom architecture* based on the experimental setup of Bluvstein et al. [9]. Zoned architectures divide the hardware layout into distinct regions, typically a *storage zone* for idle qubits and an *entanglement zone* where active two-qubit operations are performed. These can be complemented by additional zones, e.g., for readout or a reservoir of atoms to compensate for losses [10, 17]. Atoms are initially loaded into dense, static traps in the storage zone using SLMs. AODs are then used to shuttle atoms to and from the entanglement zone where global Rydberg beams perform CZ operations on selected atom pairs [25, 29]. In particular, the distance between atoms is chosen such that r_{int} and r_{restr} only include the paired atom to perform a two-qubit gate. While this avoids parallelism constraints due to blocking, it also necessitates atom rearrangements before every two-qubit layer, increasing the shuttling overhead.

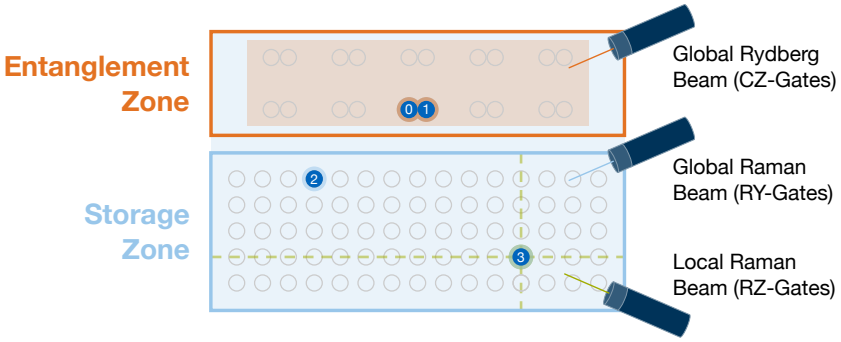


Fig. 11: Schematic of a zoned neutral atom architecture illustrating global and local operations on atoms (blue).

Example 16 In Figure 11 an illustration of a two-zone architecture is given. The entanglement zone consists of trap pairs, with qubits placed in adjacent traps (here Q_0 and Q_1) perform a CZ interaction during the global Rydberg beam. Qubits Q_2 and Q_3 are placed in the storage zone, unaffected by the Rydberg laser.

Because only atoms in the entanglement zone are affected by Rydberg excitations, all other atoms remain coherent and shielded from unintended interactions. Multiple two-qubit operations can thus be executed in parallel in a spatially isolated manner.

The compilation problem here focuses on mapping circuit operations to physical movements constrained by AOD limitations, due to the shuttling-only structure of the architecture. During a *rearrangement step*, qubits can be transferred using AODs under the constraints reviewed in Section 4.1

4.3 Proposed Solutions

Here, we present two approaches to address the compilation challenges in neutral atom quantum computing, focusing separately on hybrid and zoned architectures. These solutions consider the physical constraints, most importantly, gate restriction due to r_{restr} and shuttling parallelism due to the AOD constraints. The aim is to optimize the approximate success probability of Equation (1).

4.3.1 Hybrid Architectures

The routing problem for hybrid architectures is dominated by the choice between `SWAP`-gate insertion and AOD-shuttling. To balance this gate-based and shuttling-based routing, a hybrid compilation pipeline composed of five key stages was proposed, illustrated in Figure 12 and discussed in more detail in [89]:

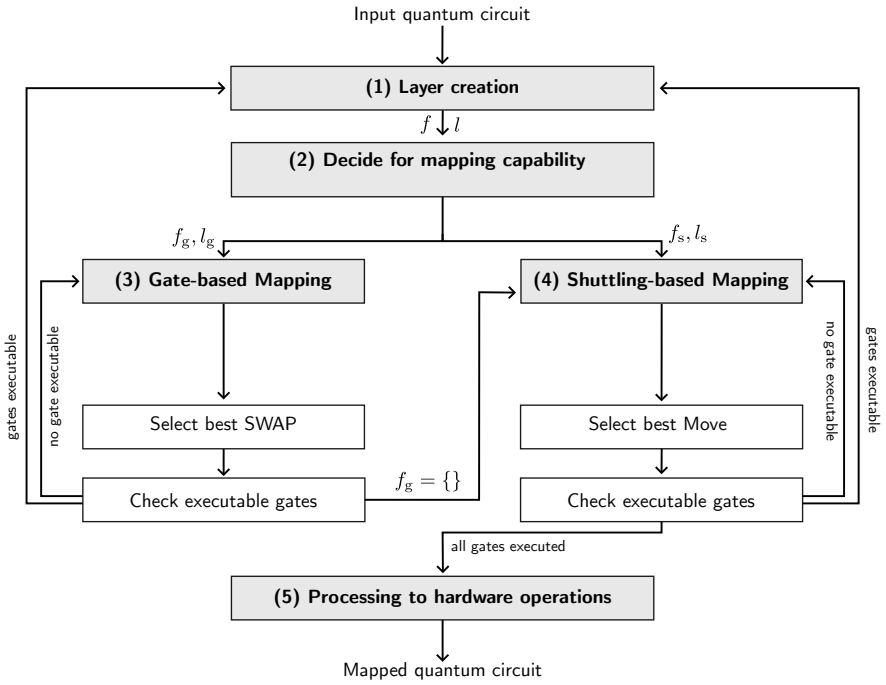


Fig. 12: Resulting hybrid mapping process, adapted from [89].

(1) **Layer Creation:** Iterate over the input circuit and identify executable gates based on gate commutativity. This defines a frontier layer f and a lookahead layer l, where the lookahead contains gates which become executable once f is empty.
(2) **Mapping Capability Decision:** Here, the costs for `SWAP`s and shuttling operations are estimated based on the arrangement and Euclidean distance of the atoms on the grid. The cost function is based on the approximate success probability to balance between gate fidelities and shuttling time. Based on that, the gates are assigned to f_g, l_g or f_s, l_s where the subscripts indicate the mapping and routing capability (gate-based s and shuttling-based s).
(3) **Gate-based Mapping:** Similar to Li et al. [57] and Baker et al. [2] all possible `SWAP` gate candidates are compared to each other using a cost function. The cost is adapted to NAQC, including distance, lookahead weight w_l, and a time-based decay factor λ^t accounting for r_{restr}.
(4) **Shuttling-based Mapping:** Similarly, possible move candidates are determined and evaluated using a cost function. The cost combines distance reduction and parallelism penalties due to AOD constraints.
(5) **Translation to Hardware Operations:** This layer decomposes `SWAP`s into native CZ and single-qubit gates and schedules AOD movements adhering, again, to the AOD constraints.

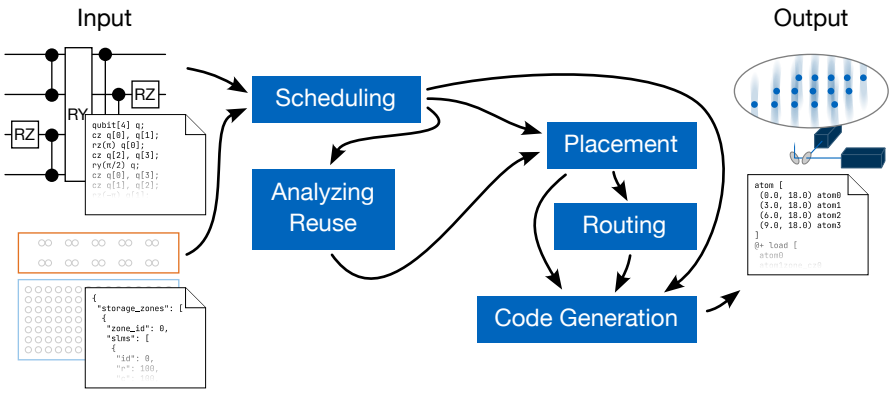


Fig. 13: The compilation flow of the routing-aware compiler introduced in [95]. The compiler takes a quantum circuit as input and produces a sequence of target-specific instructions.

As the cost functions take gate fidelities and shuttling speeds as input, the compilation will adapt to possibly device-specific characteristics. This way, the framework selects operations that are either more cost-effective or more parallelizable, depending on the scenario, providing a general and flexible compilation for hybrid architectures. Further details and implementation are provided in [89].

4.3.2 Zoned Architectures

To manage the rearrangement complexity in zoned architectures, we can employ *routing-aware placement* [95]. Here, gates are assigned to different layers, and within each layer, the qubits are mapped anew such that the routing overhead is minimized. This optimization is done using an A*-based search [97].

The input circuit is split into two types of subsequent layers: (1) Layers containing only single-qubit gates, and (2) layers containing independent two-qubit gates, i.e., two-qubit gates that can be executed in parallel. For every two-qubit gate layer, a gate and an intermediate placement are computed. The *gate placement* facilitates the execution of the corresponding two-qubit gates in the entanglement zone. An *intermediate placement* returns the qubits into the storage zone, where also the single-qubit gates are executed.

The placements are retrieved following a layer-by-layer strategy and are optimized in a way that minimizes the number of rearrangement steps in the subsequent routing stage of the compilation. The fewer rearrangement steps are required, the shorter the circuit duration and the lower the idle errors. As this routing strategy never introduces additional gate overhead, the optimization of the approximate success probability of Equation (1) reduces to an optimization of t_{idle}. Since the number of rearrangement steps and the atoms' traveling distance are the main contributions to t_{idle}, the cost

function employed by the compiler is defined as the sum over the maximum distance traveled by any atom in each rearrangement step.

$$\text{cost} = \sum_{s \in \{\text{rearr. steps}\}} \max_{d \in \{\text{travel dist. in } s\}} d$$

For finding good placements, the compiler employs an A* search algorithm that explores the placement space of the atoms. Central to A* is a suitable heuristic estimating the actual cost of placements that, in this case, combines three components [97]:

(1) An **admissible component**, estimating the lower bound of the cost.
(2) An **accelerating component**, favoring placements preserving gaps for later atoms, accelerating the search towards a good placement of all atoms.
(3) A **look-ahead component**, encouraging placements that reduce cost in future layers.

This method allows routing-aware placement to efficiently navigate an exponential placement space, minimizing the total number of rearrangement steps and, hence, the quantum circuit duration.

4.4 Discussion

Quantum circuit compilation for neutral atoms (NAQC) is significantly more complex than for static platforms like superconducting qubits due to the increased search space. Unlike fixed connectivity in superconducting chips, NAQC introduces a dynamic qubit layout via atom rearrangement, which expands the compilation problem beyond logical `SWAP`-insertion. NAQC compilation demands architecture-aware, constraint-respecting strategies, including physical constraints such as non-crossing, ghost spots, and row/column preservation of AOD rearrangements.

The best compilation strategy depends heavily on the architecture. Hybrid systems must choose between inserting `SWAP`s or shuttling atoms, balancing fidelity and latency. Zoned architectures prioritize minimizing rearrangement steps between storage and entanglement zones, where heuristic placement and scheduling (e.g., A*) can be very useful.

Finally, there have been alternative approaches to NAQC compilation for both the hybrid architecture [2, 38, 59, 75] and the zoned architecture [23, 60, 94, 101, 103]. Also, novel compilation schemes using neutral atom-specific capabilities have been proposed, such as ”flying ancillas” [104], where entanglement is mediated by additional ancillary qubits, and the idea of continuous placement of atoms instead of regular grids [62]. Continuous improvements of existing approaches and development of new methods are necessary to keep track with the rapid development of the hardware.

5 Trapped Ions

Trapped ions are among the most promising physical platforms for quantum computing. Sharing several desirable features with neutral atoms, such as their identical nature and the long coherence times, trapped-ion systems currently achieve the highest two-qubit gate fidelities of any quantum computing architecture [14, 22, 34]. Simultaneously, they suffer from scalability issues with one of the possible paths to large-scale computation, namely *Quantum Charge Coupled Devices*(QCCDs), discussed below.

5.1 Background

Trapped-ion qubits are realized using two-level quantum systems within the ions, typically based on the ions' electronic states [14, 96]. The most common encodings include hyperfine qubits [3, 8], optical qubits [79], and Zeeman-encoding based on an energy splitting induced by an external magnetic field [84]. Initialization can be achieved via optical pumping: a laser selectively excites transitions from all but one qubit state, causing the ion population to accumulate in the non-coupled state with fidelities exceeding 99.9% [12]. Measurement is performed by coupling laser light to only one of the qubit states; if the ion is in that state, it fluoresces, while the absence of fluorescence indicates the ion is in the other state [73]. This method allows for measurement fidelities exceeding 99.99% [14].

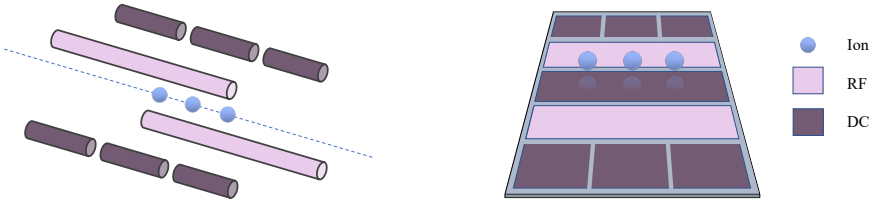
Universal *single-qubit rotations* are implemented using stimulated Raman transitions (for hyperfine qubits) or electric quadrupole transitions (for optical qubits). These operations are modeled using the Jaynes-Cummings Hamiltonian under the rotating wave approximation and are expressed as rotations on the Bloch sphere, allowing arbitrary control of the qubit state [14].

Two-qubit entangling gates rely on the Mølmer–Sørensen (MS) interaction [69], which exploits shared vibrational modes within a trap. This scheme supports *all-to-all* connectivity in an ion-chain and achieves the highest known entangling-fidelities [6, 71].

Ions are confined using RF (radio-frequency) and DC electric fields within so-called Paul traps, which can be fabricated either as three-dimensional structures or as planar surface-electrode traps [14].

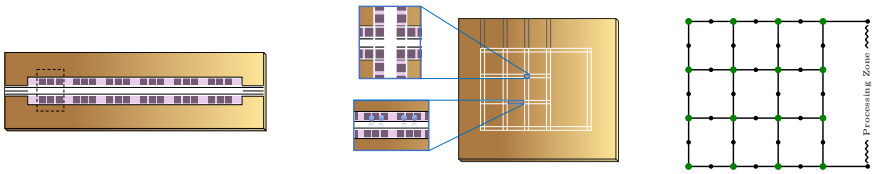
Example 17 Figure 14 illustrates Paul traps in both 3D and 2D (surface) configurations. The ions (blue) are confined by carefully shaped electric fields produced by the RF (pink) and DC (purple) control elements. Gates can be realized by focused lasers shining onto the chain either axially or orthogonally to address single ions.

While single linear traps are sufficient for small-scale devices, increasing the number of ions to more than tens or below hundreds of ions within one chain is not feasible. Among other consequences, overcrowding an ion chain leads to slower gate speeds $R_{\text{gate}} \sim 1/\sqrt{N}$ and increased motional mode crowding. These limitations



(a) 3D Paul trap (b) Surface trap

Fig. 14: An illustration of two possible realizations of a Paul trap. A combination of RF and DC electric fields produced by control electronics (pink and purple) confines the ions (blue) at their current position. The figure is adapted from [91].



(a) Linear QCCD architecture (b) 2D QCCD architecture (c) Graph abstraction

Fig. 15: An illustration of two possible QCCD architectures and a corresponding graph abstraction, adapted from [90].

motivate scalable trap architectures, where multiple individual traps are combined while ion manipulation remains independently within a single trap.

5.1.1 Quantum Charge Coupled Device (QCCD)

To enable scalable computation, the *Quantum Charge Coupled Device* (QCCD) architecture was proposed by Kielpinski et al. [45]. Similarly to the zoned neutral atom architecture, this design segments the trap into specialized functional zones: *processing zones* (PZs) for high-fidelity gate operations, *memory zones* (MZs) for storing idle qubits isolated from noise and control lasers, *measurement zones* for rapid and accurate state readout, and *loading zones* for introducing new ions. Advanced fabrication techniques, including CMOS-compatible processes, have enabled the realization of compact and scalable QCCD devices with integrated control elements.

Since ions can be moved (shuttled) throughout a trap, the ions move between the different zones as needed for the desired quantum computation [36, 72].

Example 18 Figure 15a shows a linear QCCD architecture, where each site (six purple squares) can confine and shuttle ion chains using segmented control electrodes.

To increase flexibility, linear zones can be interconnected via junctions (e.g., T-, Y-, or X-junctions) to form 2D architectures [7]. This allows parallel ion transport and circumvents the bottleneck of swap operations within a single linear trap.

Example 19 Figure 15b displays a possible 2D QCCD layout. Memory and processing zones are arranged in a grid interconnected by X-junctions, enabling scalable ion movement between zones.

While ion transport is a key advantage of trapped-ion systems, it incurs both physical and architectural costs. Linear shuttling is relatively fast, but junction crossings are slower and contribute significantly to overall latency [7, 111]. During motion, ions acquire motional energy and must be cooled, typically using Doppler and sideband cooling techniques [12], to prevent decoherence. This further increases the necessary time to move ions, making time the main optimization focus.

In the next section, we explore how quantum circuits are compiled to such QCCD architectures while optimizing for fast circuit execution.

5.2 Problem Motivation & Description

Executing quantum algorithms on trapped-ion quantum computers requires precise and efficient control of ion movement within a QCCD architecture. In this context, the considered problem is to determine a time-optimal shuttling schedule for executing a given quantum circuit on a QCCD-based architecture.

As previously discussed, the architecture consists of multiple different zones interconnected by a system of linear traps and junctions [45]. Each ion chain holds multiple qubits and acts as a single transportable entity. In the following, we focus on an architecture consisting of a grid-type memory zone, which is connected to a linear processing zone. Furthermore, we assume that the memory zone is shielded as much as possible, meaning no lasers exist to manipulate the ions. As a result, ions or ion chains can not swap places directly with each other. This means that ions may block each other, which necessitates careful coordination of the ions' movement. To work with the discussed architecture, we abstract its design as a graph.

Definition 5 Consider a graph $G = (V, E)$ (note that this is different from the coupling graph from Section 2.1.2) representing the memory zone and the interface to the processing zone. Nodes V consist of major nodes (junctions) and minor nodes (intermediate linear segments). Edges $E = \{e_0, \ldots, e_k\}$ denote all positions available to ion chains, including special inbound and outbound edges for entering and exiting the processing zone. A set $C = \{i_0, \ldots, i_l\}$ contains all ion chains relevant to the circuit.

Example 20 In the QCCD layout shown in Figure 15b, memory zones are graph-abstracted with major nodes forming a 4×4 grid, connected by edges. Minor nodes split regions between junctions, and a processing zone is connected via designated inbound and outbound edges. The corresponding graph is shown in Figure 15c.

To model the system dynamically, time is discretized into individual steps. At each time step, the state of all ion chains is described by their positions on the graph. Transitions between time steps capture movement, where no more than one ion chain may pass through a given junction at a time. This constraint ensures physical realism while maintaining abstraction from hardware-specific timing details.

Simultaneously, the quantum circuit dictates temporal constraints by specifying which ion chains must be present in the processing zone at each point. The required sequence of gate operations imposes a corresponding sequence S of ion chains derived from the circuit structure. According to that sequence, ions must move to the outbound edge to enter the processing zone, perform their operations, and return via the inbound edge.

Given these constraints, the problem addressed can be formally described as:

- **Given:** A graph layout G representing the memory zone and processing zone interface, and a sequence S derived from a quantum circuit.
- **Goal:** Determine a schedule with the minimum number of time steps $\hat{T}$ to execute all required shuttling operations for the circuit.

Because cooling and movement control infrastructure remain difficult to scale, minimizing the duration and number of shuttling operations is crucial for future large-scale devices. This motivates the search for optimized, hardware-agnostic shuttling strategies that respect architectural and operational constraints while minimizing circuit execution time.

5.3 Proposed Solution

To address the shuttling problem introduced above, two complementary methods have been proposed: an *exact solution* based on Boolean satisfiability (SAT) [91] and a *scalable heuristic* tailored to the topology of QCCD architectures [90]. The exact SAT-based method ensures optimality, while the heuristic enables efficient shuttling on larger-scale devices.

5.3.1 Exact Solution

The problem of finding a time-optimal shuttling schedule for a given quantum circuit can be formulated as a classical combinatorial optimization problem. This is a similar reasoning as for the optimal approach as the compilation for superconducting chips from Section 3.3. Although complex, this type of problem can be tackled using SAT solvers, which have become powerful tools in fields such as logic synthesis, verification, and test generation.

To apply SAT to the shuttling problem, we reduce the optimization problem to a sequence of decision problems. Specifically, we ask: ”Can the circuit be executed with a shuttling schedule of T time steps?” Starting from $T = 1$, we increment T

until a satisfiable solution is found. The first such T is guaranteed to be the minimal solution $\hat{T}$.

The problem is encoded as a SAT instance Φ, with variables representing the position and movement of ion chains at each time step, and constraints ensuring correctness, such as exclusivity of ion positions, valid transitions across junctions, and adherence to the circuit-derived sequence S [91].

Definition 6 Let Φ be a Boolean formula over a set of variables. The *Boolean satisfiability* problem is to determine if there exists an assignment of truth values such that Φ evaluates to true.

Definition 7 The Boolean variables employed to describe the position and movement of ion chains can be expressed as $x^t_{e,i} \in \{0, 1\}$ with $0 \leq t \leq T$, $i \in C$, and $e \in E$. Each such variable represents whether there is an ion chain i present on site e at time step t. The value 0 ("false") means absent, whereas 1 ("true") means present.

With this representation, further constraints can be specified to capture the full dynamics of the system. Using these constraints, an equation can be formulated, which can then be given to a SAT-solver to find a satisfiable solution in a minimal number of time steps.

While SAT is NP-complete, modern solvers are capable of solving large-scale instances efficiently. Initial work has demonstrated the potential of SAT in quantum computing contexts [108]. This work continues this trend by applying SAT solving to the domain of trapped-ion circuit compilation.

While this approach yields shuttling sequences that are optimal in the number of timesteps, the problem becomes infeasible for larger graphs. Also, in dense memory zones, many ion chains may block each other, making exact shuttling impractical. To enable scalability, another approach is necessary.

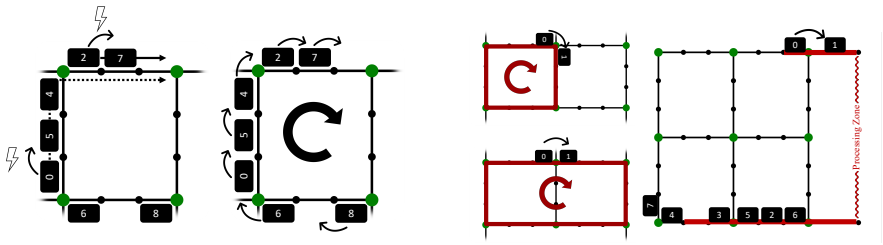
5.3.2 Heuristic and Scalable Solution

To scale to larger QCCD architectures, we also propose a heuristic based on cycle-based shuttling. The core idea is to resolve conflicts of chains blocking each other by forming cycles along chains.

A *cycle* is defined as a closed path in the graph representation of the memory zone. If a desired path is blocked, a cycle encompassing the target and blocking chains can be constructed. Moving all chains one edge forward on this cycle avoids collisions and shuttles the target ion in the right direction.

Example 21 In Figure 16a, chains c_0 and c_2 are blocked by c_5 and c_7. By forming a cycle around the blocked paths and rotating all chains, c_0 and c_2 can proceed without conflict (Figure 16b).

For grid-based QCCD layouts composed of X-junctions, cycles can be formed using rectangular regions. Vertical moves require single-rectangle cycles, while horizontal moves span two rectangles.



(a) Blocked paths (b) Cycle (c) In memory (d) Through processing

Fig. 16: Illustration of individual cycle construction and cycles across different zones. Composed and adapted from [90].

Example 22 Figure 16c illustrates two cases: (top) c_0 is blocked vertically by c_1, resolved with a single-rectangle cycle; (bottom) c_0 is blocked horizontally, resolved with a two-rectangle cycle.

Cycle-based movement is efficient because all involved junctions can be used in parallel, requiring only a single time step per full rotation. Combined, all memory zone movements can be expressed as sequences of such cycle operations.

Processing zones are accessed via a one-way pass as illustrated in Figure 16d. Direct cycles are avoided to prevent unintended chain movements. Instead, a chain is routed through the processing zone to a free edge in the memory zone.

Example 23 In Figure 16d, c_0 needs access to the processing zone, but the inbound edge is occupied by c_1. A search through the processing zone identifies a path to an unoccupied memory edge for exit.

Combining this memory zone and processing zone shuttling strategies, an arbitrary quantum circuit can be compiled to hardware QCCD operations, and efficient shuttling schedules can be created.

5.4 Discussion

Quantum circuit compilation for trapped-ion platforms, particularly with QCCD architectures, differs fundamentally from that of superconducting or neutral atom systems. Unlike static connectivity models, trapped ions require coordinated physical movement of qubits along the QCCD graph. In QCCD devices, the time overhead of shuttling ion chains between memory and processing zones makes temporal scheduling a central challenge.

These approaches demonstrate how SAT solvers can be leveraged to compute exact ion transport schedules, and the abstraction of QCCD devices as graph layouts, along with strategies such as cycle-based routing, enable us to tackle the scheduling constraints involved in compiling for large-scale devices. Together, these exact

and heuristic approaches provide a foundation for scalable and constraint-aware compilation tailored to the demands of trapped-ion devices.

6 Alternative Platforms

While superconducting qubits, trapped ions, and neutral atoms currently concentrate a large part of the experimental efforts in quantum computing, a diverse set of alternative hardware platforms is actively being explored. These systems often feature fundamentally different physical principles, control mechanisms, and architectural constraints, each requiring specialized compilation approaches tailored to their native operations, connectivity, and noise characteristics. Many methods discussed above, such as SAT-based encoding and efficient heuristics based on A^* search, could be promising candidates to develop optimal and efficient compilation software for these alternative platforms.

Photonic and Measurement-Based Quantum Computing
Photonic quantum computing [48, 93] leverages single photons—typically represented in specific optical modes—as qubits. Quantum gates can be implemented using linear optical elements such as beam splitters, phase shifters, and interferometers, which perform unitary transformations on the photon modes. Measurement-based models [83], such as those relying on cluster states, shift the computational effort to the generation of entangled resource states followed by adaptive measurements, posing distinct compilation challenges related to resource optimization and feed-forward logic [112, 113].

Quantum Dots and Spin Shuttling Architectures In architectures based on spin qubits, qubits are encoded into the spin states of electrons in quantum dots [21]. Operations are implemented through controlled exchange interactions with proposals that introduce qubit shuttling along 1D or 2D arrays [52] to dynamically alter the connectivity graph, imposing unique temporal and spatial constraints on circuit compilation [24].

Silicon-Based Quantum Computing Silicon is a promising platform owing to its compatibility with existing semiconductor fabrication technologies. Qubits can be implemented using donor-based approaches, such as phosphorus atoms. Silicon qubits benefit from long coherence times—especially in isotopically purified silicon-28—and high-fidelity gate operations. However, achieving high connectivity and scaling remains difficult. The compilation process must accommodate layout constraints and the typically sparse or dynamic connectivity, as well as limited parallelism in control [20].

Together, each platform introduces its own architectural primitives and noise models, underscoring the importance of hardware-aware quantum compilation strategies beyond those developed for the mainstream platforms.

7 Conclusion

As quantum hardware platforms grow in scale and maturity, they continue to face fundamental resource constraints such as limited qubit connectivity, restricted mobility, and fidelity limitations.

Different hardware platforms impose distinct challenges that require tailored compilation strategies. Superconducting qubits necessitate careful `SWAP` gate insertion due to static coupling maps, with symbolic and heuristic methods balancing optimality and scalability. Neutral atom systems demand dynamic qubit layout management, addressing rearrangement constraints like non-crossing and zone-based operations. Trapped-ion platforms with QCCD architectures introduce temporal scheduling challenges tied to ion shuttling, solvable via both SAT-based and heuristic techniques.

This work demonstrates that hardware-specific strategies are key to efficient quantum compilation. All the presented methods above are available in open-source as part of the *Munich Quantum Toolkit* [109] (MQT), giving easy access to potential users and developers.

While superconducting qubits, trapped ions, and neutral atoms currently dominate experimental efforts, a range of alternative quantum hardware platforms is rapidly emerging. Systems such as photonic and measurement-based quantum computing, quantum dots with spin shuttling, and silicon-based qubits introduce fundamentally different operational principles and constraints. Extending the techniques discussed in this chapter—including SAT-based encodings and efficient heuristic methods like A^* search—offers a promising direction for developing compilation tools tailored to these next-generation platforms.

References

[1] M. Amy et al. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 32(6):818–830, 2013.

[2] J. M. Baker et al. Exploiting Long-Distance Interactions and Tolerating Atom Loss in Neutral Atom Quantum Architectures. In *Int'l Symposium on Computer Architecture*, pages 818–831, 2021.

[3] C. J. Ballance et al. High-Fidelity Quantum Logic Gates Using Trapped-Ion Hyperfine Qubits. *Physical Review Letters*, 117(6):060504, 2016.

[4] K. Barnes et al. Assembly and coherent control of a register of nuclear spin qubits. *Nature Communications*, 13(1):2779, 2022.

[5] D. Barredo et al. An atom-by-atom assembler of defect-free arbitrary two-dimensional atomic arrays. *Science*, 354(6315):1021–1023, 2016.

[6] J. Benhelm et al. Towards fault-tolerant quantum computing with trapped ions. *Nature Physics*, 4(6):463–466, 2008.

[7] R. B. Blakestad et al. High-Fidelity Transport of Trapped-Ion Qubits through an x-junction trap array. *Physical Review Letters*, 102(15):153002, 2009.

[8] B. B. Blinov et al. Quantum Computing with Trapped Ion Hyperfine Qubits. *Quantum Information Processing*, 3(1):45–59, 2004.

[9] D. Bluvstein et al. A quantum processor based on coherent transport of entangled atom arrays. *Nature*, 604(7906):451–456, 2022.

[10] D. Bluvstein et al. Logical quantum processor based on reconfigurable atom arrays. *Nature*:1–3, 2023.

[11] A. Botea et al. On the complexity of quantum circuit compilation. In *Int'l Symp. on Combinatorial Search*, 2018.

[12] M. F. Brandl. A quantum von neumann architecture for large-scale quantum computing, 2017. arXiv: `1702.02583`.

[13] S. Bravyi et al. The future of quantum computing with superconducting qubits. *Journal of Applied Physics*, 132(16):160902, 2022.

[14] C. D. Bruzewicz et al. Trapped-ion quantum computing: progress and challenges. *Applied Physics Reviews*, 6(2):021314, 2019.

[15] J.-L. Brylinski et al. Universal quantum gates, 2001. arXiv: `quant-ph/0108062`.

[16] L. Burgholzer et al. Limiting the Search Space in Optimal Quantum Circuit Mapping. In *Asia and South Pacific Design Automation Conf.* Pages 466–471, 2022. arXiv: `2112.00045`.

[17] N.-C. Chiu et al. Continuous operation of a coherent 3,000-qubit system, 2025. arXiv: `2506.20660`.

[18] F. T. Chong et al. Programming languages and compiler design for realistic quantum hardware. *Nature*, 549(7671):180–187, 2017.

[19] A. Cowtan et al. On the Qubit Routing Problem. In W. van Dam et al., editors, *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, volume 135 of *Leibniz International Pro-*

ceedings in Informatics (LIPIcs), 5:1–5:32, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. eprint: 1902.08091.

[20] O. Crawford et al. Compilation and scaling strategies for a silicon quantum processor with sparse two-dimensional connectivity. *npj Quantum Information*, 9(1):1–11, 2023.

[21] M. De Smet et al. High-fidelity single-spin shuttling in silicon. *Nature Nanotechnology*:1–7, 2025.

[22] S. Debnath et al. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536(7614):63–66, 2016.

[23] E. Decker. Arctic: A Field Programmable Quantum Array Scheduling Technique, 2024. arXiv: 2405.06183.

[24] P. Escofet et al. Compilation Techniques for Spin Qubits in a Shuttling Bus Architecture, 2025. arXiv: 2502.06263.

[25] S. J. Evered et al. High-fidelity parallel entangling gates on a neutral-atom quantum computer. *Nature*, 622(7982):268–272, 2023.

[26] T. F. Gallagher. *Rydberg Atoms*. Cambridge Monographs on Atomic, Molecular and Chemical Physics. Cambridge University Press, Cambridge, 1994.

[27] Y. Ge et al. Quantum Circuit Synthesis and Compilation Optimization: Overview and Prospects, 2024. arXiv: 2407.00736.

[28] B. Giles et al. Exact synthesis of multiqubit Clifford+T circuits. *Phys. Rev. A*, 87(3):032332, 2013.

[29] G. Giudici et al. Fast entangling gates for Rydberg atoms via resonant dipole-dipole interaction, 2024. arXiv: 2411.05073.

[30] T. M. Graham et al. Rydberg-mediated entanglement in a two-dimensional neutral atom qubit array. *Physical Review Letters*, 123(23):230501, 2019.

[31] R. Grimm et al. Optical dipole traps for neutral atoms. In B. Bederson et al., editors. Volume 42, Advances in Atomic, Molecular, and Optical Physics, pages 95–170. Academic Press, 2000.

[32] F. Gyger et al. Continuous operation of large-scale atom arrays in optical lattices. *Phys. Rev. Res.*, 6:033104, 3, 2024.

[33] T. Häner et al. A software methodology for compiling quantum programs. *Quantum Science and Technology*, 3(2):020501, 2018.

[34] T. P. Harty et al. High-fidelity trapped-ion quantum logic using near-field microwaves. *Physical Review Letters*, 117(14), 2016.

[35] L. Henriet et al. Quantum computing with neutral atoms. *Quantum*, 4:327, 2020.

[36] W. K. Hensinger. Quantum computer based on shuttling trapped ions. *Nature*, 592(7853):190–191, 2021.

[37] H.-L. Huang et al. Superconducting quantum computing: a review. *Science China Information Sciences*, 63(8):180501, 2020.

[38] Y. Huang et al. DasAtom: A Divide-and-Shuttle Atom Approach to Quantum Circuit Transformation, 2025. arXiv: 2409.03185.

[39] L. Isenhower et al. Demonstration of a neutral atom controlled-NOT quantum gate. *Physical Review Letters*, 104(1):010503, 2010.

[40] T. Itoko et al. Optimization of quantum circuit mapping using gate transformation and commutation. *Integration*, 70:43–50, 1, 2020.

[41] S. Jandura et al. Time-Optimal Two- and Three-Qubit Gates for Rydberg Atoms. *Quantum*, 6:712, 2022.

[42] A. Javadi-Abhari et al. Quantum computing with Qiskit, 2024. arXiv: 2405.08810.

[43] A. JavadiAbhari et al. ScaffCC: a framework for compilation and analysis of quantum computing programs. In *Conference on Computing Frontiers*, pages 1–10, 2014.

[44] A. Jenkins et al. Ytterbium nuclear-spin qubits in an optical tweezer array. *Physical Review X*, 12(2):021027, 2022.

[45] D. Kielpinski et al. Architecture for a large-scale ion-trap quantum computer. *Nature*, 417(6890):709–711, 2002.

[46] A. Kissinger et al. PyZX: Large Scale Automated Diagrammatic Reasoning. *Electronic Proceedings in Theoretical Computer Science*, 318:229–241, 2020. arXiv: 1904.04735.

[47] A. Y. Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191, 1997.

[48] P. Kok et al. Linear optical quantum computing with photonic qubits. *Reviews of Modern Physics*, 79(1):135–174, 2007.

[49] P. Krantz et al. A quantum engineer's guide to superconducting qubits. *Applied Physics Reviews*, 6:021318, 2019.

[50] F. Kreppel et al. Quantum Circuit Compiler for a Shuttling-Based Trapped-Ion Quantum Computer. *Quantum*, 7:1176, 2023.

[51] J. Kunasaikaran et al. A Framework for the Design and Realization of Alternative Superconducting Quantum Architectures, 2024. arXiv: 2305.07052.

[52] M. Künne et al. The SpinBus architecture for scaling spin qubits with electron shuttling. *Nature Communications*, 15(1):4977, 2024.

[53] L. Lao et al. Timing and Resource-Aware Mapping of Quantum Circuits to Superconducting Processors. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 41(2):359–371, 2022.

[54] H. Levine. Quantum Simulation and Quantum Information Processing with Programmable Rydberg Atom Arrays. https://dash.harvard.edu/handle/1/37368497, 2021.

[55] H. Levine et al. Parallel implementation of high-fidelity multiqubit gates with neutral atoms. *Physical Review Letters*, 123(17):170503, 2019.

[56] H. Levine et al. Parallel Implementation of High-Fidelity Multiqubit Gates with Neutral Atoms. *Phys. Rev. Lett.*, 123(17):170503, 2019.

[57] G. Li et al. Tackling the qubit mapping problem for NISQ-era quantum devices. In *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 2019.

[58] G. Li et al. Tackling the qubit mapping problem for NISQ-era quantum devices. In *Int'l Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1001–1014.

[59] Y. Li et al. Timing-Aware Qubit Mapping and Gate Scheduling Adapted to Neutral Atom Quantum Computing. *IEEE Trans. on CAD of Integrated Circuits and Systems*:1–1, 2023.

[60] W.-H. Lin et al. Reuse-Aware Compilation for Zoned Quantum Architectures Based on Neutral Atoms. In *Int'l Symposium on High-Performance Computer Architecture*, 2025. arXiv: 2411.11784.

[61] W.-H. Lin et al. Scalable Optimal Layout Synthesis for NISQ Quantum Processors. In *Design Automation Conf.* Pages 1–6, 2023.

[62] J. Ludmir et al. Parallax: A Compiler for Neutral Atom Quantum Computers under Hardware Constraints. In *Int'l Conf. for High Performance Computing*, SC '24, pages 1–17, Atlanta, GA, USA. IEEE Press, 2024.

[63] S. Ma et al. Universal gate operations on nuclear spin qubits in an optical tweezer array of ^{171}Yb atoms. *Physical Review X*, 12(2):021028, 2022.

[64] M. Maronese et al. Quantum compiling. In S. S. Iyengar et al., editors, *Quantum Computing Environments*, pages 39–74. Springer International Publishing, Cham, 2022.

[65] D. Maslov et al. Quantum circuit simplification and level compaction. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(3):436–444, 2008.

[66] D. Maslov. On the advantages of using relative phase Toffolis with an application to multiple control Toffoli optimization. *Phys. Rev. A*, 93(2):022311, 10, 2016.

[67] D. M. Miller et al. Elementary quantum gate realizations for multiple-control Toffoli gates. In *Int'l Symp. on Multi-Valued Logic*, 2011.

[68] Z. K. Minev et al. Qiskit Metal: An Open-Source Framework for Quantum Device Design & Analysis, 2021.

[69] K. Mølmer et al. Multiparticle entanglement of hot trapped ions. *Physical Review Letters*, 82(9):1835–1838, 1999.

[70] M. Morgado et al. Quantum simulation and computing with Rydberg-interacting qubits. *AVS Quantum Science*, 3(2):023501, 2021.

[71] S. A. Moses et al. A race track trapped-ion quantum processor, 2023. arXiv: 2305.03828.

[72] P. Murali et al. Architecting noisy intermediate-scale trapped ion quantum computers. In *Int'l Symposium on Computer Architecture*, ISCA '20, pages 529–542, Virtual Event. IEEE Press, 2020.

[73] A. H. Myerson et al. High-Fidelity Readout of Trapped-Ion Qubits. *Physical Review Letters*, 100(20):200502, 2008.

[74] M. A. Norcia et al. Iterative assembly of ^{171}Yb atom arrays with cavity-enhanced optical lattices. *PRX Quantum*, 5:030316, 3, 2024.

[75] N. Nottingham et al. Decomposing and Routing Quantum Circuits Under Constraints for Neutral Atom Architectures, 2023. arXiv: 2307.14996.

[76] S. Park et al. A fast and scalable qubit-mapping method for noisy intermediate-scale quantum computers. In *Design Automation Conf.* Pages 13–18, 2022.

[77] T. Peham et al. On Optimal Subarchitectures for Quantum Circuit Mapping. *ACM Transactions on Quantum Computing*, 2023. arXiv: 2210.09321.

[78] A. Peres. Reversible logic and quantum computers. *Physical Review A*, 32(6):3266–3276, 1985.

[79] I. Pogorelov et al. Compact Ion-Trap Quantum Computing Demonstrator. *PRX Quantum*, 2(2):020343, 2021.

[80] J. Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, 2018.

[81] S. Pucher et al. Fine-Structure Qubit Encoded in Metastable Strontium Trapped in an Optical Lattice, 2024. arXiv: 2401.11054.

[82] S. Rasmussen et al. Superconducting circuit companion—an introduction with worked examples. *PRX Quantum*, 2(4):040204, 2021.

[83] R. Raussendorf et al. Measurement-based quantum computation on cluster states. *Physical Review A*, 68(2):022312, 2003.

[84] T. Ruster et al. A long-lived Zeeman trapped-ion qubit. *Applied Physics B*, 122(10):254, 2016.

[85] M. Saffman. Quantum computing with atomic qubits and Rydberg interactions: progress and challenges. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 49(20):202001, 2016.

[86] M. Saffman et al. Quantum information with Rydberg atoms. *Reviews of Modern Physics*, 82(3):2313–2363, 2010.

[87] M. Saffman. Quantum computing with neutral atoms. *National Science Review*, 6(1):24–25, 2019.

[88] L. Schmid et al. Computational capabilities and compiler development for neutral atom quantum processors—connecting tool developers and hardware experts. *Quantum Science and Technology*, 9(3):033001, 2024.

[89] L. Schmid et al. Hybrid Circuit Mapping: Leveraging the Full Spectrum of Computational Capabilities of Neutral Atom Quantum Computers. In *Design Automation Conf.* 2024. arXiv: 2311.14164.

[90] D. Schoenberger et al. Shuttling for Scalable Trapped-Ion Quantum Computers. *IEEE Trans. on CAD of Integrated Circuits and Systems*:1–1, 2024.

[91] D. Schoenberger et al. Using Boolean Satisfiability for Exact Shuttling in Trapped-Ion Quantum Computers. In *Asia and South Pacific Design Automation Conf.* Pages 127–133, 2024.

[92] I. Siddiqi. Engineering high-coherence superconducting qubits. *Nature Reviews Materials*, 6(10):875–891, 2021.

[93] S. Slussarenko et al. Photonic quantum information processing: A concise review. *Applied Physics Reviews*, 6(4):041303, 2019.

[94] Y. Stade et al. An Abstract Model and Efficient Routing for Logical Entangling Gates on Zoned Neutral Atom Architectures. In *Int'l Conf. on Quantum Computing and Engineering*, 2024. arXiv: 2405.08068.

[95] Y. Stade et al. Routing-aware placement for zoned neutral atom-based quantum computing, 2025. arXiv: 2505.22715.

[96] A. Steane et al. Quantum Computing with Trapped Ions, Atoms and Light. *Fortschritte der Physik*, 48(9-11):839–858, 2000.

[97] S. J. Stefan Edelkamp. Cost-Algebraic Heuristic Search.

[98] K. Svore et al. A layered software architecture for quantum computing design tools. *Computer*, 39(1):74–83, 2006.

[99] B. Tan et al. Optimal layout synthesis for quantum computing. In *Int'l Conf. on CAD*, pages 1–9, 2020.

[100] B. Tan et al. Optimality Study of Existing Quantum Computing Layout Synthesis Tools. *IEEE Transactions on Computers*, 70(9):1363–1373, 2021.

[101] D. B. Tan et al. Compiling Quantum Circuits for Dynamically Field-P rogrammable Neutral Atoms Array Processors. *Quantum*, 8:1281, 2024.

[102] G. Unnikrishnan et al. Coherent Control of the Fine-Structure Qubit in a Single Alkaline-Earth Atom, 2024. arXiv: 2401.10679.

[103] H. Wang et al. Atomique: A Quantum Compiler for Reconfigurable Neutral Atom Arrays. In *Int'l Symposium on Computer Architecture*, pages 293–309, 2024. arXiv: 2311.15123 [quant-ph].

[104] H. Wang et al. Q-Pilot: Field Programmable Quantum Array Compilation with Flying Ancillas. In *Design Automation Conf.* 2024. arXiv: 2311.16190.

[105] G. Wendin. Quantum information processing with superconducting circuits: a review. *Reports on Progress in Physics*, 80(10):106001, 2017.

[106] R. Wille et al. Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In *Design Automation Conf.* Pages 1–6, 2019.

[107] R. Wille et al. Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In *Design Automation Conf.* 2, 2019.

[108] R. Wille et al. MQT QMAP: efficient quantum circuit mapping. *CoRR*, abs/2301.11935, 2023. arXiv: 2301.11935.

[109] R. Wille et al. The MQT Handbook: A Summary of Design Automation Tools and Software for Quantum Computing. In *IEEE International Conference on Quantum Software (QSW)*, 2024. arXiv: 2405.17543. A live version of this document is available at https://mqt.readthedocs.io.

[110] K. Wintersperger et al. Neutral atom quantum computing hardware: performance and end-user perspective. *EPJ Quantum Technology*, 10(1):32, 2023.

[111] K. Wright et al. Reliable transport through a microfabricated X-junction surface-electrode ion trap. *New Journal of Physics*, 15(3):033004, 2013.

[112] H. Zhang et al. OneQ: A Compilation Framework for Photonic One-Way Quantum Computation. In *Int'l Symposium on Computer Architecture*, pages 1–14, 2023.

[113] F. Zilk et al. A compiler for universal photonic quantum computers. In *2022 IEEE/ACM Third International Workshop on Quantum Computing Software (QCS)*, pages 57–67, 2022. arXiv: 2210.09251.

[114] A. Zulehner et al. An efficient methodology for mapping quantum circuits to the IBM QX architectures. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2019.