TOPICAL REVIEW • OPEN ACCESS

Computational capabilities and compiler development for neutral atom quantum processors—connecting tool developers and hardware experts

To cite this article: Ludwig Schmid *et al* 2024 *Quantum Sci. Technol.* **9** 033001

IOP Publishing Quantum Sci. Technol. 9 (2024) 033001 https://doi.org/10.1088/2058-9565/ad33ac

Quantum Science and Technology

OPEN ACCESS

RECEIVED
22 September 2023

REVISED
21 February 2024

ACCEPTED FOR PUBLICATION
13 March 2024

PUBLISHED
3 April 2024

TOPICAL REVIEW

Computational capabilities and compiler development for neutral atom quantum processors—connecting tool developers and hardware experts

Ludwig Schmid[1,*], **David F Locher**[2,3], **Manuel Rispler**[2,3], **Sebastian Blatt**[4,5,6], **Johannes Zeiher**[4,5,6], **Markus Müller**[2,3] and **Robert Wille**[1,7]

[1] Chair for Design Automation, Technical University of Munich, 80333 Munich, Germany
[2] Institute for Quantum Information, RWTH Aachen University, 52056 Aachen, Germany
[3] Peter Grünberg Institute, Theoretical Nanoelectronics, Forschungszentrum Jülich, 52425 Jülich, Germany
[4] Faculty of Physics, Ludwig-Maximilians-Universität, 80799 Munich, Germany
[5] Max-Planck-Institut für Quantenoptik, 85748 Garching, Germany
[6] Munich Center for Quantum Science and Technology (MCQST), 80799 Munich, Germany
[7] Software Competence Center Hagenberg GmbH (SCCH), 4232 Hagenberg im Mühlkreis, Austria
[*] Author to whom any correspondence should be addressed.

E-mail: ludwig.s.schmid@tum.de, d.locher@fz-juelich.de, rispler@physik.rwth-aachen.de, sebastian.blatt@mpq.mpg.de, johannes.zeiher@mpq.mpg.de, markus.mueller@fz-juelich.de and robert.wille@tum.de

Keywords: quantum computing, neutral atoms, design automation, compiler, software tool development

Abstract

Neutral Atom Quantum Computing (NAQC) emerges as a promising hardware platform primarily due to its long coherence times and scalability. Additionally, NAQC offers computational advantages encompassing potential long-range connectivity, native multi-qubit gate support, and the ability to physically rearrange qubits with high fidelity. However, for the successful operation of a NAQC processor, one additionally requires new software tools to translate high-level algorithmic descriptions into a hardware executable representation, taking maximal advantage of the hardware capabilities. Realizing new software tools requires a close connection between tool developers and hardware experts to ensure that the corresponding software tools obey the corresponding physical constraints. This work aims to provide a basis to establish this connection by investigating the broad spectrum of capabilities intrinsic to the NAQC platform and its implications on the compilation process. To this end, we first review the physical background of NAQC and derive how it affects the overall compilation process by formulating suitable constraints and figures of merit. We then provide a summary of the compilation process and discuss currently available software tools in this overview. Finally, we present selected case studies and employ the discussed figures of merit to evaluate the different capabilities of NAQC and compare them between two hardware setups.

1. Introduction

To achieve computational advantages with *Quantum Computers* (QC), large-scale, high-fidelity qubit entanglement is required, posing a technologically challenging problem. In recent years, qubit systems based on *Neutral Atoms* (NA) [1, 2] in combination with Rydberg interactions have established themselves as a promising candidate, due to their ability to perform high-fidelity long-range gates [3–5], native multi-qubit gates [4–7], and physical atom shuttling [8, 9], combined with their scalability [10–12].

However, to fully harness these capabilities, it becomes essential to devise hardware-specific optimization techniques and software tools. In particular, this includes *compilation*, i.e. translating high-level algorithmic descriptions into a low-level representation of operations that can be executed on the hardware, obeying given physical constraints, and optimizing for specific figures of merit. Manual optimization becomes infeasible as system sizes scale, necessitating automated processes and comprehensive toolkits to establish a

complete compilation pipeline. While a multitude of frameworks is available for other hardware platforms, such as *superconducting* chips [13–22], or *trapped ions* [23–27], the landscape of compiler tools tailored to NA-specific hardware constraints [28–36] is still less developed.

Existing solutions for NAs often address specific compilation subproblems or make assumptions about a particular hardware configuration, failing to fully leverage the expansive capabilities of the NAQC platform. To properly ensure that corresponding compilers and tools obey physical constraints and optimize for hardware-specific figures of merit, a close connection between tool developers and hardware experts is required. The aim is to create valuable and high-quality compilation software that can leverage and take advantage of the full range of computational capabilities intrinsic to the NAQC platform.

This work aims to provide a basis to establish this connection by furnishing a comprehensive overview of software compilation for the NAQC platform and laying the groundwork for potential directions in compiler development geared explicitly toward adaptive, hardware-aware compilation strategies. We discuss important figures of merit and employ them to evaluate the different capabilities of NAs, as well as compare them between two different hardware setups.

To achieve this goal, first, we establish a connection from physics to computer science by reviewing the physical background of NAs and translating the physical principles and processes to a hardware-aware but more abstract problem formulation suitable for tool developers. The correspondingly resulting 'take home-messages' for tool developers are then summarized in the form of optimization constraints and figures of merit in self-contained boxes, suitable résumés for individuals who are already familiar with NAs. In this discussion, we particularly focus on the NA-specific capabilities of long-range connectivity, native multi-qubit gates, and the possibility of implementing MOVE operations on the qubits, using shuttling.

Secondly, we discuss how these new capabilities impact the compilation task, give a comprehensive overview of the full range of the compilation possibilities, and contextualize currently available software within this overview. This discussion gives a possibility to structure the current progress of compilation development and aids potential tool developers in identifying unsolved subproblems and automation tasks.

Finally, we present selected case studies and error analysis to provide an overview of the current state of the art of hardware-aware compilation for NAQC. This provides insights for tool developers regarding suitable figures of merit and optimization metrics depending on available hardware properties. Concurrently, this gives hardware experts the means to estimate the impact of their hardware configuration on the final compilation output and could potentially aid in devising forthcoming hardware arrangements, prioritizing hardware attributes that yield the most encouraging outcomes.

Based on these three main contributions, we provide the basis for a successful connection between tool developers and hardware experts for physical realization, which is necessary for the future development of hardware-aware compilation tools, taking full advantage of the broad spectrum of capabilities of the NAQC platform.

The remainder of this work is structured as follows: In section 2, we discuss the specific compilation aspects addressed in this work. We define key terms and compilation steps, such as synthesis, mapping, and routing, at an abstract and hardware-independent level. In section 3, we focus on the NAQC platform. Here, we explore its distinctive capabilities and characteristics relevant to compilation. Each subsection covers a particular capability and discusses its underlying physical principles and mechanisms. This is followed by an abstract formulation tailored to the computer science community, summarized in self-contained boxes referring to the three previously introduced compilation steps. In section 4, we investigate how these novel capabilities influence the compilation process, outlining a potential overview and illustrating the overall compilation problem in an overview figure. Referring to this framework, section 5 reviews existing software and discusses what capabilities they focus on, giving a comprehensive overview of currently available software for NAQC. Section 6 offers selected case studies that showcase how the NA platform's various capabilities and hardware parameters influence the computation results.

2. Compilation

In its broadest sense, the term 'compilation' refers to translating a high-level, abstract description of a quantum algorithm into a lower-level representation of operations suitable for hardware execution. This is typically achieved by employing multiple layers of software, each designed to address specific subroutines. The collective arrangement of these layers is commonly known as a *compilation tool-chain* [37]. Given the diverse range of tasks involved and the intersection between computer science- and physics-related terminology, various terms, including the term 'compilation' itself, can vary depending on the context. To avoid ambiguity, we provide a brief review and definitions of different subroutines considered in this work.

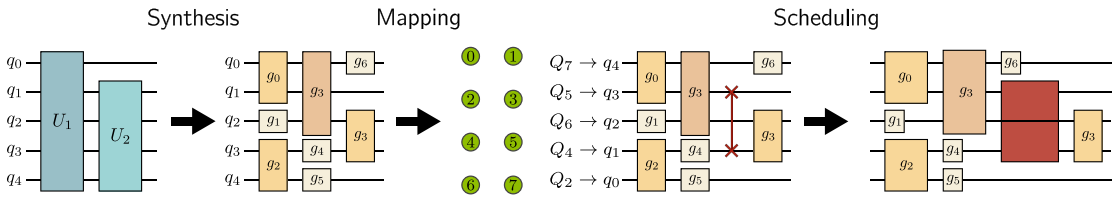

Figure 1. Illustration of the three steps for platform-dependent compilation. In the synthesis step, general operations and unitaries are decomposed into the native gate set Σ_{native}. During the mapping step, the circuit qubits q_i are assigned to physical hardware qubits Q_i, and necessary SWAP or MOVE operations are introduced to satisfy connectivity constraints. Finally, in the scheduling step, gate times and restrictions on parallelism are considered. In practice, these steps are often performed simultaneously as a single step rather than sequentially.

2.1. Compilation subroutines

The subroutines can be divided into three major categories, depending on their abstraction level and their specificity to a particular hardware configuration.

- *Platform-independent compilation*
  High-level optimization techniques can be applied irrespective of the underlying platform. These techniques include for example loop unrolling and function inlining [38], to substitute or simplify e.g. for-loops and function calls on a high level similar to classical compilers. Another possibility are optimization techniques on the gate level such as gate commutation rules.
- *Platform-dependent compilation*
  These subroutines account for a given hardware platform's specific capabilities and constraints, such as superconducting chips, trapped ions, NAs, or photonic quantum computing. The output of these subroutines comprises a sequence of platform-specific instructions, which is still agnostic of the physical hardware setup.
- *Hardware-dependent compilation*
  This translation layer is tailored to each hardware configuration, converting abstract quantum operations into hardware-specific instructions that can be directly executed on the quantum processing unit. It is occasionally referred to as *firmware* to underscore its close proximity to the underlying hardware.

This work focuses on addressing the subroutines of platform-dependent compilation, particularly the NAQC platform. Specifically, we assume that all hardware- and platform-independent optimizations have already been conducted, and our objective is to generate a collection of hardware-oriented operations without delving into discussions related to direct hardware control through electric signals or pulse-level intricacies. This leads to three primary objectives.

1. *Synthesis* entails decomposing abstract gates into operations compatible with the provided platform and is therefore also often referred to as *decomposition*.
2. *Mapping* involves spatially arranging the gates by assigning circuit qubits to corresponding hardware qubits and inserting SWAP or MOVE operations to fulfill connectivity constraints.
3. *Scheduling* corresponds to the temporal arrangement of the gates to satisfy the dependencies and consider the parallelism constraints inherent to the platform or the hardware.

In the following, we present concise and abstract definitions of these three steps tailored to the context of this work. An illustration is shown in figure 1.

2.1.1. Synthesis

Within the framework of the *quantum circuit model*, every non-dissipative quantum computation $U \in \mathbb{C}^{2^n \times 2^n}$ can be expressed as a finite sequence of quantum operations $g \in \mathbb{C}^{2^m \times 2^m}$ known as quantum gates, denoted by reversible unitary transformations. Here, n (m) denotes the number of qubits upon which the circuit (gate) acts. According to the Solovay-Kitaev theorem [39], given a *universal gate set* Σ^{univ} with a discrete number of gates, an approximate decomposition up to an arbitrarily small error can always be obtained. Furthermore, this can be done efficiently in terms of the number of gates. In contrast, the *native gate* set Σ^{native} characterizes the feasible operations that can be performed on the quantum state using a specific hardware platform or setup. For universal computing, it is necessary that Σ^{native} is also a universal gate set.

Definition 1 Synthesis). Given a quantum computation $U \in \mathbb{C}^{2^n \times 2^n}$ and the native platform gate set Σ^{native}, *synthesis* is the task to find a gate sequence

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

with all $g_0, \ldots g_N \in \Sigma^{\text{native}}$ and $U = \tilde{U}$ up to some small error.

2.1.2. Mapping

Each gate operates on a subset of the *circuit qubits* denoted as $\mathbf{Q} = \{q_0, \ldots, q_{n-1}\}$. For instance, $g_0(q_0, q_1)$ indicates that the first gate of the computation acts on the circuit qubits q_0 and q_1. These gates must be implemented using the available physical *hardware qubits* represented by $\mathbf{P} = \{Q_0, \ldots, Q_{n-1}\}$. Without loss of generality, we assume that the number of circuit and physical qubits is the same. A common challenge encountered in current hardware platforms is limited connectivity, which is described by a *coupling graph* $G = (\mathbf{P}, \mathbf{E})$. In this graph, the nodes correspond to the physical qubits, while the edges $\mathbf{E}$ indicate the qubits capable of interacting with each other.

To execute a gate $g(Q_i, Q_j)$ with $(Q_i, Q_j) \notin \mathbf{E}$ (that is, the qubits are not directly connected), it is necessary to adjust the coupling graph to establish the required connectivity. In most platforms, connectivity is closely related to the physical locations of the qubits. Therefore, two commonly employed techniques are SWAP and MOVE operations. The SWAP(Q_i, Q_j) operation exchanges the positions of the qubits Q_i and Q_j, resulting in a modification of their labels in the coupling graph. On the other hand, the MOVE(Q_i) operation relocates qubit Q_i to a different position, consequently reassigning the associated edges.

In superconducting (SC) hardware, these operations are typically performed at the virtual level, acting on $\mathbf{Q}$, and require, for example, three controlled-NOT (CX) gates to implement a SWAP operation. On the contrary, for other platforms, such as trapped ions or NAs, it may be possible to physically move and swap the corresponding atoms or ions, directly affecting the hardware qubits $\mathbf{P}$.

Definition 2 (Mapping). Given a quantum circuit $U = g_{N-1} \circ \cdots \circ g_0$ on circuit qubits $\mathbf{Q}$ and a hardware configuration with physical qubits $\mathbf{P}$ and coupling map $G(\mathbf{P}, \mathbf{E})$. The task of *mapping* is to find a bijective function $f : \mathbf{Q} \to \mathbf{P}$ and an insertion of MOVE and SWAP operations such as

$$U = \ldots \circ \text{MOVE}\,(q_i) \circ \text{SWAP}\,\left(q_j, q_k\right) \circ g\left(q_i, q_j\right) \circ \ldots$$

such that for each gate $g(q_i, q_j)$ all inter-qubit connections are fulfilled, i.e. $(f(q_i), f(q_j)) \in \mathbf{E}$.

It should be noted, that this graph-based approach to mapping can only represent a first approximation. In general and, in particular, for multi-qubit gates, additional constraints such as gate direction and the geometric arrangement of the qubits can impose additional constraints.

2.1.3. Scheduling

While mapping primarily concerns the spatial arrangement of gates on qubits to satisfy connectivity constraints, we must also consider the temporal positioning of the gates. Subject to commutation rules, gates acting on the same qubit must be executed in a specific order. This aspect can be abstracted by transforming the gate sequence $U = g_{N-1} \circ \cdots \circ g_0$ into a *Directed Acyclic Graph* (DAG) denoted D. In this DAG, the nodes correspond to quantum gates, while the incoming and outgoing edges correspond to the qubits on which the gates operate. The direction of the edges reflects the sequential execution order of the gates.

When two gates act on disjoint sets of qubits, lacking any common path in the DAG, they can generally be executed in parallel. However, the execution of gates in parallel may face additional constraints imposed by hardware limitations, such as the availability of control channels for qubit control, or platform-specific restrictions on gate operations, such as cross-talk effects.

Definition 3 (Scheduling). Given a quantum circuit U and its corresponding DAG representation D, the objective of *scheduling* is to determine the optimal timing for the gates to be executed while preserving the integrity of the DAG up to commutation rules.

Due to decoherence, the primary goal during the scheduling phase is typically to maximize parallelism, thereby minimizing the overall execution time of the circuit.

In the next section, we delve into a comprehensive study of the NAQC platform, exploring its unique computational capabilities compared to other platforms. In particular, we analyze how the physical principles and processes of the NAs can be formulated on an abstract level regarding optimization constraints and figures of merit, focusing on the three compilation steps of synthesis, mapping, and scheduling.

3. The neutral atom quantum computing platform

Building on continuous efforts in using NAs for atomic clocks and analog quantum simulations, several recent experimental and theoretical breakthroughs have allowed this platform to evolve into a promising candidate for scalable digital quantum computing [1–4, 40–44]. This section provides an overview of the fundamental physical principles essential for employing NAs in quantum computing and the resulting additional capabilities. Based on that, we then translate these physical principles and processes to a hardware-aware but more abstract problem formulation, establishing a connection between physics and computer science and, hence, hardware experts and tool developers. This results in 'take-home messages' for tool developers, which are provided in terms of dedicated boxes covering optimization constraints as well as figures of merit and entailing everything relevant for compiler development. Individuals with prior knowledge of NAs and/or compilation for quantum hardware may only check these boxes or bypass the corresponding sections completely. In the following, we first show that the NAQC is a suitable candidate for universal quantum computation. We discuss in depth the NA capabilities and error sources, each entailed by one of the aforementioned summary boxes focusing on the three hardware-dependent compilation steps of synthesis, mapping, and routing, extended by hardware and error discussions.

3.1. Platform requirements

For the NAQC to be suitable as a quantum computing platform, one requires multiple properties, which are commonly known as DiVincenzo's criteria [45], namely:

1. A scalable physical system of qubits.
2. The ability to initialize the state of the qubits to a simple fiducial state.
3. A universal set of quantum gates.
4. Long relevant coherence times, much longer than the gate operation time.
5. A qubit-specific measurement capability.

The following outlines that all those requirements are fulfilled for NAs trapped in optical tweezer arrays or optical lattices.

1. System and qubits: NAs can be stored in optical dipole traps [46], which are typically formed by far-detuned lasers interacting with the atomic dipole moment. Examples include optical lattices, which are stationary patterns of light that emerge from the interference of multiple laser beams [47], or optical tweezer arrays, which consist of many individual tightly focused and individually controllable laser beams arranged in one, two, or three dimensions [48]. In state-of-the-art optical tweezer arrays, atoms are loaded stochastically and can then be dynamically rearranged in desired configurations [11, 12], with the possibility of dynamically reloading atoms from a reservoir to compensate for occasional loss of atoms [49, 50]. It is possible to laser-cool the atoms within the traps to their motional ground states [51]. Commonly used atom species in such experiments are alkali atoms, e.g. Rb, Cs, or alkaline-earth-like atoms, such as Sr or Yb. Specific internal electronic states of the atoms serve as the qubit states $|0\rangle$ and $|1\rangle$. Both states are typically encoded in two long-lived states of the atom, such as two hyperfine states in alkali atoms [4, 52], a nuclear spin [53–55], low-energy singlet and triplet states [56, 57], fine-strucure [58, 59], or circular Rydberg states [60] in alkaline-earth (like) atoms. Due to their different properties, also dual-species atom arrays have been proposed and demonstrated [61].

2. Initialization, and 3. Quantum gates: Transitions between electronic states can be precisely controlled by applying laser pulses to the atoms. Such pulses are used to initialize qubits in well-defined initial states, e.g. $|0\rangle$, and to perform arbitrary single-qubit gates. Universal quantum computing also demands controlled two-qubit gates. To achieve this, atoms can be temporarily excited to Rydberg states, which are electronic states with very large principal quantum numbers. Atoms in Rydberg states exhibit large polarizabilities, and as a consequence, two Rydberg atoms interact via dipole-dipole interactions [62]. By coupling one of the qubit states, e.g. $|1\rangle$, to a Rydberg state $|r\rangle$, one obtains an effective interaction between two atoms in the state $|1\rangle$ [40, 63]. Depending on the exact settings, interaction characteristics can vary from dipolar to van der Waals [2]. This mechanism can be used to engineer two-qubit or multi-qubit gates.

4. Coherence times: The two primary error sources of trapped atoms during idling are dephasing and amplitude damping ($|1\rangle \mapsto |0\rangle$). These processes cause the off-diagonal terms in the density matrix ρ of a qubit to decay on a time scale T_2^*, and the matrix element ρ_{11} to decay on a time scale T_1. Both time scales can reach up to the order of several seconds for NAs in optical tweezers [5, 56] and are characteristic for the specific qubit implementation, e.g. magnetically insensitive clock states with very long lifetimes. Controlled

phase gates can be performed in timescales of the order of 100 ns–500 ns [64], depending on the physical details, which is orders of magnitude faster than the decoherence time of such systems.

5. Measurements: Finally, the qubit states of the atoms can be measured by performing fluorescence imaging, that is, driving transitions between one of the qubit levels and an auxiliary electronic level with a laser and imaging the emitted photons with a camera.

3.2. Computational capabilities

In addition to fulfilling DiVincenzo's criteria, NAs offer a broad spectrum of capabilities that can be employed to perform quantum computations. In the following, these capabilities are discussed in more detail. This is done by first discussing the corresponding capabilities' basic physical principles and processes, followed by self-contained boxes, summarizing the resulting constraints and optimization figures of merit on a more abstract level. Therefore, tool developers can refer only to the boxes for the development of NA-specific compilers.

3.2.1. Single-qubit gates

Single-qubit gates are realized with lasers that drive Rabi oscillations between the qubit states $|0\rangle$ and $|1\rangle$. The respective Hamiltonian reads

$$\frac{H_1(t)}{\hbar} = \frac{\Omega(t)}{2}|0\rangle\langle 1| + \frac{\Omega^*(t)}{2}|1\rangle\langle 0| - \Delta(t)\,|1\rangle\langle 1|, \tag{1}$$

where $\Omega(t)$ and $\Delta(t)$ are the effective Rabi frequency and detuning, respectively. Depending on the precise qubit encoding, the transitions might be either single- or two-photon transitions. In the latter case, the Hamiltonian in equation (1) is obtained after adiabatic elimination of an intermediate level. Depending on the experimental setup, some laser beams might only be available as global beams [3], simultaneously illuminating many atoms. However, single-qubit addressing can also be realized using beams focused on individual atoms [34, 65].

3.2.2. Two-qubit gates

To execute two-qubit gates, typically, one of the qubit states, e.g. $|1\rangle$, is coupled to a *Rydberg state* $|r\rangle$ using a laser with Rabi frequency Ω_r and detuning Δ_r. Two atoms in the same Rydberg state and separated sufficiently far from each other interact via the van der Waals interaction

$$V_\mathrm{vdW}(d) = \frac{C_6}{d^6}, \tag{2}$$

where C_6 is a state-specific constant and d the inter-qubit distance. The Hamiltonian describing two atoms illuminated by a global Rydberg laser, therefore, reads

$$\frac{H_2(t)}{\hbar} = \sum_{i=1}^{2}\left(\frac{\Omega_\mathrm{r}(t)}{2}|1\rangle\langle r|_i + \frac{\Omega_\mathrm{r}^*(t)}{2}|r\rangle\langle 1|_i - \Delta_\mathrm{r}(t)\,|r\rangle\langle r|_i\right) + \frac{V_\mathrm{vdW}}{\hbar}|r\rangle\langle r|_1 \otimes |r\rangle\langle r|_2. \tag{3}$$

The van der Waals interaction imposes an energy penalty to promote two nearby atoms together to a Rydberg state. Consequently, laser excitation of one atom to a Rydberg state is impossible if another Rydberg atom is close by. This so-called *Rydberg blockade* can be utilized to engineer state-dependent interactions and, thus, two-qubit gates between two atoms. The first proposal by Jaksch *et al* [63] requires a sequence of locally addressed laser pulses and was realized for the first time in 2010 with neutral Rb atoms [66, 67]. More recent two-qubit gate sequences require only global addressing of both involved qubits and are also faster than the original scheme [4]. The qubits must be located sufficiently close to each other so that the interaction is strong enough for the Rydberg blockade to hold. From the condition $\hbar\Omega_\mathrm{r} \simeq V_\mathrm{vdW}(r_\mathrm{b})$ follows a blockade radius

$$r_\mathrm{b} \simeq \left(\frac{C_6}{\hbar\Omega_\mathrm{r}}\right)^{1/6}. \tag{4}$$

This length scale gives an estimate of how far two atoms participating in a two-qubit gate can be separated at most. Since the blockade radius can be much larger than the distance between neighboring atoms, two-qubit gates can be applied to pairs of atoms that are not nearest neighbors, resulting in higher inter-qubit connectivity.

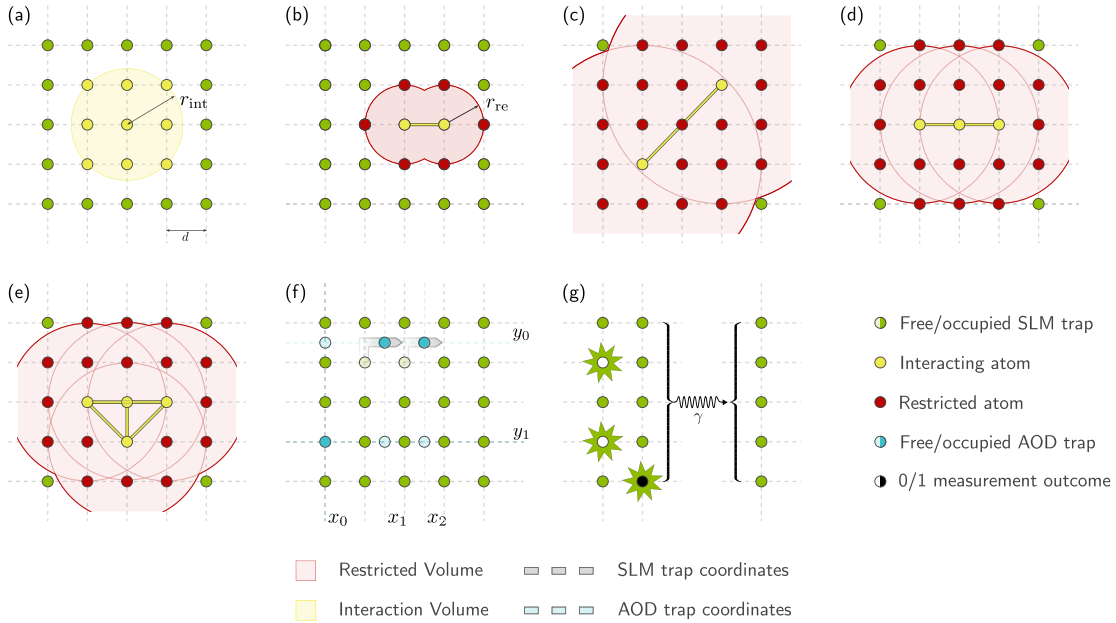

Figure 2. Capabilities of the NAQC platform. In this setup, atoms are arranged on a regular grid of Spatial Light Modulator (SLM) traps, with a fixed distance denoted as d. (a) Rydberg blockade interaction: within a specific interaction zone of radius $r_{\text{int}} = \sqrt{2}d$, depicted in yellow, interacting gates can be performed to all qubits within this range. (b) Two-qubit gate: a gate can be applied between neighboring qubits but restricts the simultaneous execution of other entangling gates on nearby atoms. The restriction volume, represented by a red sphere of radius $r_{\text{re}} = d$, indicates this restriction. The interaction radius $r_{\text{int}} = d$ is not explicitly shown to simplify the illustration. (c) Long-Range interactions: for gates with larger interaction radii, the restriction zones ($r_{\text{re}} = 3d \geqslant r_{\text{int}} = 2\sqrt{2}d$) also expand, resulting in more restricted atoms. (d) CCZ gate with a line arrangement of the qubits. According to [4] it is sufficient if the central atom interacts with both the outer qubits, resulting in a minimal interaction radius of $r_{\text{int}} = d$. The restriction radius is illustrated examplatory as $r_{\text{re}} = 2d$. (e) CCCZ gate: in this case, we require that all four gate qubits must be within the interaction radius $r_{\text{int}} = r_{\text{re}} = 2d$ of every other qubit, according to equation (8). (f) Shuttling operation: dynamic Acousto-Optic Deflector (AOD) traps (blue) enable the movement of atoms within the same column (x) or row coordinate (y) simultaneously. The procedure addresses certain constraints discussed in section 3.2.4. (g) Additional NA capabilities, useful for future fault-tolerant computations. For example, mid-circuit measurements and possible inter-photonic connections.

Synthesis: (One- and two-qubit gates)
Assuming local addressability of all qubits, this results in a native gate set $\Sigma^{\text{univ}} = \{R, CZ\}$ containing arbitrary single-qubit rotations R and the two-qubit CZ gate to synthesize a given algorithm.

Nevertheless, synthesis for NAs is hard, as the exact synthesis steps required depend heavily on the chosen atom species and qubit encoding. Furthermore, one has to distinguish between laser pulses available as global or locally addressable beams. A first approach, leveraging SC synthesis tools with additional post-processing was demonstrated by Nottingham *et al* [32].

Mapping: Two-qubit gates can be executed between any pair of qubits sufficiently close for the blockade to take effect. This constraint can be described by a constant *interaction radius* denoted as r_{int}, where a gate $g(Q_i, Q_j)$ can be performed if

$$d\left(Q_i, Q_j\right) \leqslant r_{\text{int}}, \tag{5}$$

where d is the Cartesian distance. This is illustrated in figure 2(a).

We can define a coupling graph $\mathbf{C} = (V, E)$, with

$$E = \left\{ \left(Q_i, Q_j\right) | d\left(Q_i, Q_j\right) \leqslant r_{\text{int}} \right\}. \tag{6}$$

With this coupling graph, common mapper tools of the SC platform [13, 14, 16, 18, 19] can be partially employed for the NAQC platform.

However, during the two-qubit gate sequence, both involved qubits are promoted to the Rydberg state and, therefore, could interact with other nearby atoms in Rydberg states, resulting in unwanted detrimental

cross-talk. While the strong Rydberg interaction is indispensable for achieving high-fidelity two-qubit gates, it also imposes limitations on other atoms in close proximity, preventing them from simultaneously performing similar gates.

Scheduling: To establish the constraint on simultaneous gate executions, we use the concept of a *restriction radius* denoted as r_{re}. This parameter represents the minimum distance required between two atoms in the Rydberg state to prevent undesirable cross-talk. It is worth noting that, in general, $r_{\mathrm{re}} \geqslant r_{\mathrm{int}}$, as cross-talk may arise even at distances where a gate interaction may not yet be feasible.

Thus, for two gates $g(Q_i, Q_j)$ and $g'(Q_a, Q_b)$ to be executed in parallel, the following condition must be satisfied:

$$d(Q_i, Q_a), d(Q_i, Q_b), d(Q_j, Q_a), d(Q_j, Q_b) > r_{\mathrm{re}}. \tag{7}$$

If the distances between qubits involved in gates are too small, gates must be executed in sequential order, possibly increasing the execution time of the circuit. This limitation is commonly known as *blocking*; however, to avoid any potential confusion with the concept of Rydberg blockade, we use the term *restriction*. We refer to the region surrounding a gate, affected by this phenomenon, as the *restriction volume*.

A simple visual example with different radii is illustrated in figures 2(b) and (c), where the restriction volume and the corresponding restricted atoms are colored in red.

The interdependence of r_{int} and r_{re} on the strength of the Rydberg blockade r_{b} allows us to establish a relationship between them as $r_{\mathrm{re}} = k \cdot r_{\mathrm{int}}$, where $k \geqslant 1$ is referred to as *blocking factor*. This gives rise to an interesting trade-off between higher connectivity achieved with a larger r_{int}, which, in turn, leads to larger restriction volumes and consequently reduces the number of parallel gate executions. This phenomenon has been investigated by Baker *et al* [33] and is further discussed in greater detail with additional case studies and error analysis in section 6.

3.2.3. Multi-qubit gates

In NA quantum processors, it is also possible to apply native multi-qubit gates to sets of atoms.

Analogous to the case of two-qubit gates, the atoms must be located within the blockade radius of each other such that the atoms interact when excited to Rydberg states. There exist multiple theoretical proposals to implement multi-qubit gates such as $\mathrm{C}_k\mathrm{Z}$ and CZ_k gates, up to single-qubit rotations, and also more exotic versions [5–7, 68]. Parallel applications of Toffoli (CCZ) gates to multiple sets of atoms have already been demonstrated in experiments with trapped Rb atoms [4, 5].

Synthesis: This expands the possible native gate set to $\Sigma^{\mathrm{univ}} = \{R, \mathrm{C}_k\mathrm{Z}, \mathrm{CZ}_k\}$ and, therefore, gives much more possibilities to synthesize a given circuit. First, multi-qubit gates within an algorithm do not have to be decomposed to a more basic gate set, possibly increasing gate or decoherence errors.

Furthermore, native multi-qubit gates are of great interest for the implementation of classical reversible circuits, using Toffoli and multiple controlled gates [69–72]. Those classical circuits can be used for algorithmic subtasks or the synthesis of oracles used in quantum algorithms. In this context, oracles refer to black-box functions performing the required operation. Although often used during the construction of quantum algorithms, the task of finding an adequate realization is non-trivial in general.

Secondly, instead of decomposing unitaries to one- and two-qubit gates only, also other gate combinations such as Hadamard plus Toffoli [73, 74] may be used, resulting in alternative decompositions.

Mapping: The mapping constraints for multi-qubit gates depend on the specific implementation and the corresponding laser pulses. To simplify the discussion we assume the worst case, where each qubit needs to interact with each other qubit within the gate. Therefore, to implement a multi-qubit gate $g(Q)$ acting on the set of qubits $Q = \{Q_i, \ldots, Q_j\}$, all qubits have to be within the interaction radius of each other:

$$d(Q_i, Q_j) \leqslant r_{\mathrm{int}} \, \forall i, j \in Q. \tag{8}$$

Furthermore, this implies that r_{int} places a constraint on the size and configuration of multi-qubit gates. For instance, a Toffoli gate with its three qubits aligned in a straight line necessitates $r_{\text{int}} \geqslant 2d$ as in figure 2(d), while a possible perpendicular arrangement could be achieved with $r_{\text{int}} \geqslant \sqrt{2}d$.

As mentioned, this constraint's extent can be mitigated depending on the specific gate and its implementation. For instance, in the case of the CCZ gate [4] in figure 2(d), solely a single qubit needs to be within r_{int} of the others, while among themselves, they may be situated at a greater distance. On the other hand, depending on the implementation, constraints in addition to equation (8) must be fulfilled. Taking again the example of the CCZ gate, one also has to take into consideration the geometric arrangement of the qubits. In this case, the gate requires the three qubits to sit along a straight line, while a perpendicular arrangement would not be possible [4].

Scheduling: The restriction mechanism from the two-qubit gates generalizes to the case of two multi-qubit gates by composing the minimal distance r_{re} between any two qubits of the respective gates. So, for two gates $g(Q)$ and $g'(Q')$ acting on the two-qubit sets Q, Q' we need

$$d(Q_i, Q_m) \geqslant r_{\text{re}} \quad \forall Q_i \in Q \text{and} \, \forall Q_m \in Q'. \tag{9}$$

In difference to the interaction constraint of equation (8), this cannot be relaxed, as both control and target qubits are potentially in the Rydberg state during gate execution. As a result, multi-controlled gates restrict a larger number of atoms compared to simpler gates.

The necessity for a larger interaction radius, along with the increased number of qubits, leads to an expanded restriction volume for multi-qubit gates in contrast to two-qubit gates. Nevertheless, this could potentially be compensated by a more efficient synthesis process facilitated by utilizing a larger native gate set or the faster execution of the corresponding multi-qubit gate.

3.2.4. Atom shuttling

In the context of NAQC, individual tweezers holding qubits can be dynamically moved during computation without disrupting entanglement, as demonstrated in [8, 9, 75]. This capability offers an alternative to implementing SWAP operations at the virtual level, requiring three CX gates. In addition, physically shuttling atoms to new locations provides the flexibility to establish dynamic connectivity between qubits.

One physical realization of these shuttle operations involves placing atoms intended for shuttling within a tweezer array generated using a 2D crossed *Acousto-Optic Deflector* (AOD). On the contrary, stationary qubits remain in a static tweezer array formed by employing a *Spatial Light Modulator* (SLM). A typical complete shuttling operation requires first a pick-up from a static SLM to a dynamic AOD trap. Via a controlled frequency ramp applied to the AOD, the qubit is rearranged to the destination, followed by a controlled release of the atom back into a static trap. The selection of atoms to move can be altered for subsequent maneuvers, and specific parallel moves are feasible [42, 48], with the constraints summarized in the following box.

Mapping: The AOD can be characterized by two sets of coordinates: $x_i, \ldots x_k$ and $y_a, \ldots y_c$. Each intersection (x_i, y_a) defines a potential trap where an atom can be confined. Shuttling is achieved by modifying these coordinates, effectively relocating the corresponding traps. Consequently, changing the value of a specific coordinate, say x_0, results in the simultaneous movement of all AOD-trapped atoms in the first column, with identical displacements in the same direction. This implies that not all atoms within a single AOD can be moved independently. By examining the movement vectors $\vec{m}_{i,a}$ for atoms located at coordinate (x_i, y_a), the following constraints emerge:

$$\begin{aligned} \vec{m}_{i,a} \cdot \hat{e}_x &= \vec{m}_{i,b} \cdot \hat{e}_x \quad \forall a, b \\ \vec{m}_{i,a} \cdot \hat{e}_y &= \vec{m}_{j,a} \cdot \hat{e}_y \quad \forall i, j \ , \end{aligned} \tag{10}$$

where $\hat{e}_x$ and $\hat{e}_y$ are the respective unit vectors. An illustrative example of this scenario is shown in figure 2(f), where two atoms are simultaneously shuttled row-wise. On the contrary, a third AOD-trapped atom remains stationary, corresponding to different row and column traps.

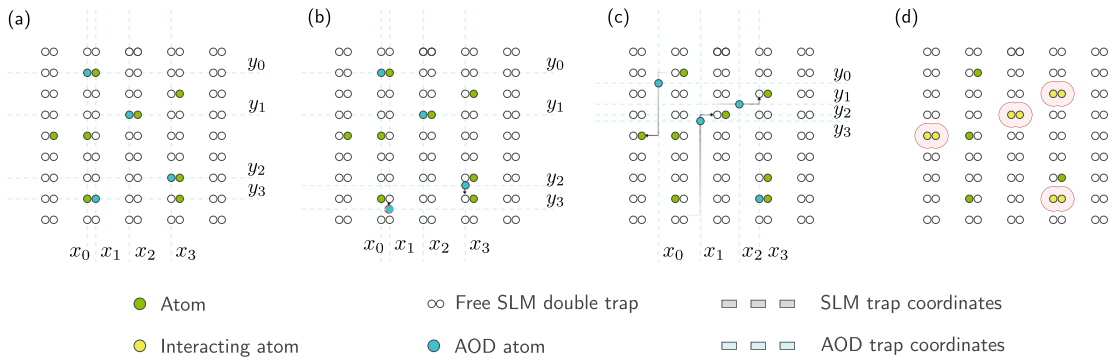

Figure 3. Neutral Atom Dynamically Field-Programmable Array (DPQA) Processor. In this architecture, atoms are strategically placed within Spatial Light Modulator (SLM) trap pairs, ensuring that the distance between them is smaller than r_{int} while maintaining an inter-pair distance larger than r_{re} to prevent mutual restrictions. Computation in a DFPA processor can be considered as the repetition of three major phases: (a) Loading phase: a subset of atoms undergoes a switching process to be rearranged using Acousto-Optic Deflectors (AODs), indicated by the AOD coordinates x and y. (b) + (c) Shuttling phase: the trapped atoms are rearranged by readjusting the AOD coordinates. To avoid conflicts due to row crossing, the atom in the lower left corner (indicated in red) must be displaced first, allowing the other atom to occupy its designated position. In particular, for y_2 to reach its destination, the y_3 row has to be moved along as the coordinates are not allowed to cross. Subsequently, the remaining atoms, and also y_3, can be shuttled to their respective locations. For more complicated shuttling operations this requires sophisticated methods to find adequate AOD movements. (d) Gate phase: an entangling CZ gate can be performed on atoms within the same pair of traps. This operation can be accomplished either with a global laser beam or individually using selective beams, as discussed in figure 2. Phases (a)–(d) are repeated iteratively until all the required gates have been executed.

As a second constraint, we observe that two coordinates are not allowed to be closer than a given minimal distance d_{min}

$$\begin{aligned} |x_i - x_j| &> d_{\text{min}} \quad \forall i,j \\ |y_a - y_b| &> d_{\text{min}} \quad \forall a,b. \end{aligned} \tag{11}$$

Otherwise, this would result in overlapping trap potentials and undefined behavior with potential atom loss. Equation (11) enforces that the coordinate lines cannot cross each other, maintaining a fixed ordering $x_0, \ldots, x_k$ throughout the process.

Scheduling: There are two possible approaches to shuttle multiple AOD-trapped atoms in parallel.

First, by spanning an AOD with multiple x or y coordinates to trap multiple atoms using a single AOD. In this scenario, all parallel movements must comply with the constraints described in equations (10) and (11). Specifically, columns (rows) can only move along the same x (y) displacement and must not cross each other.

The second case involves using multiple AODs, each with their respective coordinates. In this setup, movements from different AODs are independent of each other, allowing for maximum flexibility. The remaining constraint is to ensure that the atoms keep a minimal distance through the shuttling process.

Instead of using shuttling only as a substitute for virtual SWAP and MOVE operations, the recently discussed *Dynamically Field-programmable Qubit Arrays* (DPQA or D-FPQA) [8, 9, 30, 35] has emerged as a quantum processor design, which entirely focuses on shuttling. Compared to the previous discussion, this new computation architecture is not based on qubits placed on a regular grid. A short definition is given in the following box and described in figure 3.

Architecture: (DPQA)
In the Dynamically Field-Programmable Qubit Arrays (DPQA) setup, stationary SLM traps are arranged in groups on a grid, ensuring that the qubit distance within a single group is smaller than r_{int}, allowing gates to be applied within the group. Meanwhile, the groups themselves are placed at an inter-group distance greater than r_{re}, preventing interference between gates on different groups, as

illustrated in figure 3. This arrangement requires more space than the previous configuration, especially for larger r_{re} values.

The computation process in DPQA can be divided into three phases, assuming that all atoms start in one of the SLM traps:

1. *Loading:* Depending on the following gates to be executed, atoms that need to be shuttled are loaded into the AOD traps by activating the corresponding coordinates.
2. *Shuttling:* Modulating the AOD coordinates enables the atoms to move column/row-wise. Due to the constraints of equations (10) and (11), direct movement of atoms to their destinations may not always be possible, necessitating intermediate shuttling, as depicted in figure 3(b). Here, the left atom has to be shuttled downwards first to let the other atom reach its destination. After reaching their destinations, the atoms switch traps back to stationary SLM traps.
3. *Gate execution:* Entangling gates can be performed on qubits within the same group. This can be achieved by addressing individual atoms, possibly implementing multi-qubit gates, or using a global beam to apply the same entangling gate on all groups with more than one atom. In this phase, required single-qubit gates can also be performed by addressing the atoms individually or globally.

These three steps are iterated until all the gates in the circuit have been executed. Due to the movement dependencies of equation (10), finding suitable AOD movements is a highly non-trivial task.

An extension of the DPQA setup adds the possibility of defining separate zones for dedicated entangling, measuring and storing qubits which allows for experimentally optimized setups [8]. The routing between these zones on the other hand imposes an additional computational overhead, resulting in a trade-off situation regarding processor design similar to the discussions between gate-based and shuttling-based mapping in section 6.4.

3.2.5. Measurements

The qubit state of an atom can be measured by performing fluorescence imaging on a cycling transition between one of the computational states and an auxiliary electronic state [1]. During this process, the fluorescence light emitted by the atoms is imaged with a camera such that bright spots in a final image correspond to atoms in a specific qubit state. However, this procedure is experimentally challenging. Scattering of photons for readout heats the atoms, leading to significant atom loss from the shallow optical dipole traps without laser cooling. Measurements that preserve the atoms in the traps have been demonstrated in free space [76, 77], but might be easier to achieve by performing cavity-enhanced fluorescence imaging [78, 79]. Next to being faster and non-destructive, the latter technique has the advantage that much fewer photons must be scattered for detection, reducing atom heating and cross-talk problems at the expense of serial readout. Parallel mid-circuit measurements of qubits have also been demonstrated recently [8, 80–84].

3.2.6. Errors

Qubits in a quantum computation can suffer from *idle-errors*, incurring when the qubits are unused, or *gate errors* when imperfect quantum gates are applied. During idling, NA qubits mainly experience two error processes. First, atoms can decay from the qubit state with higher energy, commonly referred to as $|1\rangle$, to the ground state, which typically encodes the qubit $|0\rangle$ state. This *amplitude damping* causes the matrix element ρ_{11} of the density matrix ρ of a qubit to decay on a characteristic time scale T_1, called *relaxation time*. The relaxation time for NAQC is typically very large; it can be of the order of 100 s for hyperfine qubits, but even for other qubit encodings T_1 can reach several seconds [56]. Second, atoms undergo *dephasing*, mainly resulting from fluctuating external fields or drive fields. Due to both error processes, the off-diagonal elements of the qubit's density matrix decay on a time scale T_2^*. Canceling noise on slower time scales with dynamical decoupling techniques results in a modified, increased, *dephasing time* T_2. Encoding a qubit in two states insensitive to external field fluctuations yields long dephasing times that can be as high as several seconds. As a simplification to the exact error analysis, which is highly non-trivial in general, we want to define the artificial parameter of an *effective coherence time*

$$T_{\text{eff}} = \frac{T_1 T_2}{T_1 + T_2}. \tag{12}$$

If either $T_2 \ll T_1$ or vice versa, T_{eff} reduces to the respective shorter time, such that it correctly captures the typical time scale, limiting the system's coherence.

Furthermore, the application of gates can induce errors on a qubit. Besides experimental imperfections such as e.g. miscalibrations or laser intensity fluctuations, there are physical processes that limit achievable gate fidelities. In particular, two- and multi-qubit gates require atoms to be temporarily excited to Rydberg states. Those states decay quickly, either to the qubit subspace, leading to *computational errors*, or to other electronic states, resulting in *leakage errors*. Experimentally, specific pulse shapes can be utilized to minimize the effect of such errors [85, 86]. Typically, these Rydberg decay errors are combined with other gate execution errors, resulting in a gate-specific *average gate fidelity* $\mathcal{F}_g$ for a given gate g.

Unfortunately, even given the average gate fidelities of all gates in a quantum circuit, it is not possible to compute the exact final output fidelity. In general, it is even hard to establish useful thresholds, as the exact fidelity depends on complicated microscopic error characteristics and their interplay with each other. To still account for different gate fidelities and coherence times, we consider a strongly simplified error discussion in the following which should be considered only a very rough estimate of the actual errors. Nonetheless, we can consider it a first proxy criterion regarding optimization techniques during compilation.

We can summarize this discussion of physical errors into two principal error types. Decoherence errors for idle qubits and gate fidelities for executing gates. In this simplified scheme, we can make the following abstraction from physical error processes to an error measure, which requires only a small set of physically motivated parameters to get a basic but hardware-aware error estimation.

Errors: Given a quantum computation U with synthesis, mapping, and scheduling already performed, including inserted MOVE and SWAP operations, we obtain a sequence of $\tilde{N}$ operations O:

$$U = O_{N-1} \circ \cdots \circ O_{\tilde{0}}.$$

In addition, let the fidelities f of all operations and the coherence times T_1 and T_2 be given. We can define the abstract measure of *approximate success probability* P. It is defined as the product of all average gate fidelities, combined with the term for the idle error:

$$P(U) = \exp\left(-\frac{t_{\text{idle}}}{T_{\text{eff}}}\right) \prod_{i=0}^{\tilde{N}} \mathcal{F}_{O_i}, \tag{13}$$

where T_{eff} is given according to equation (12) and the *idle-time* t_{idle} describes the time in which no gate is applied to a qubit, summed over the whole register. In the case that all operations, including SWAPs and MOVEs, are realized by gates, equation (13) simplifies to

$$P(U) = \exp\left(-\frac{t_{\text{idle}}}{T_{\text{eff}}}\right) \prod_{i=0}^{N} \mathcal{F}_{g_i}, \tag{14}$$

where we only need the gate fidelities $\mathcal{F}_{(g_i)}$ for all gates $g_i \in \Sigma_{\text{native}}$ in the native gates set.

The *idle-time* can be computed given the gate execution times t and the total circuit execution time T as

$$t_{\text{idle}} = n \cdot T - \sum_{i=0}^{N} t(g_i), \tag{15}$$

after routing has been employed to compute T.

The scheme of approximate success probability provides a simple and fast-calculated proxy criterion for the occurrence of errors in quantum computation. The general concept allows comparison between operations at different abstraction levels, such as native gates or oracles, and operations performed using additional capabilities, such as SWAP gates or SWAP shuttling movements, as long as it is possible to assign an operation-specific execution time t and an average fidelity $\mathcal{F}$.

3.2.7. Fault-tolerant QC and error correction

For the long-term goal of large-scale quantum computation, it will be necessary to employ techniques from quantum fault tolerance to deal with the accumulation of noise and errors [87]. In the case of NAs, not only computational errors but also leakage errors must be corrected to achieve fault tolerance. There are proposals

to convert leakage errors into Pauli Z-errors [88] or detect and thus convert them into erasures [89–92]. Two-qubit gate designs have been proposed in which errors mainly occur in the form of detectable erasures [93, 94].

Regarding quantum error correction, most schemes require mid-circuit measurements and real-time feedback, a procedure which has been demonstrated lately with NAs [8, 83, 84] but remains challenging. Therefore, progress on measurement-free fault-tolerant quantum error correction might be promising for NAs [95–97]. Also, the possibility of using atom-photon interactions has been discussed lately [98, 99], allowing for photonic interconnects on long distances. Due to experimental demonstrations [8], current work is focused on the possibility of using the shuttling capability of NAs to prepare and manage encoded states for qLDPC [100], the surface code [101], and generalized bicycle codes [102]. Combined with the discussion on fault-tolerant realizations of non-Clifford gates [103], this paves the way towards fault-tolerant computation on the NAQC platform. This is of particular interest, as the requirements for proposed qLDPC code realizations on superconducting platforms, are still beyond current hardware capabilities [104–106].

4. Compilation for neutral atoms

Meeting DiVincenzo's criteria at the hardware level is not sufficient to achieve a fully operational quantum computer. Abstractly formulated quantum algorithms must be translated into physical operations available on the hardware and carefully coordinated to satisfy constraints and conditions imposed by the physical process or experimental setup.

In this section, we will explore how the computational capabilities and constraints of the NAQC platform, as discussed in section 3.2, impact hardware-focused compilation steps. Additionally, we will propose a set of simplified proxy criteria as figures of merit to evaluate the compilation results, enabling comparison between fundamentally different operations, such as virtual or physical swapping.

This discussion proposes potential compilation paths to address the increased number of possibilities, given the diverse quantum operation capabilities of NAs. Such efforts may aid the development of design automation and compilation software, taking full advantage of the capability spectrum of the NAQC platform. In particular, regarding the choice of figures of merit to evaluate the compilation results.

4.1. Overview

The NAQC platform introduces a novel set of capabilities for implementing gates and new degrees of freedom for conducting operations, but consequently, also additional constraints within the compilation processes. Moreover, for certain operations like shuttling, conventional figures of merit for evaluating compilation quality are no longer directly applicable. This necessitates new schemes on how a given quantum algorithm is compiled and how the efficacy of the resulting compilation can be assessed. To this end, we propose an abstract compilation overview that considers the distinct computational capabilities and appropriate figures of merit for optimization processes and evaluation in the specific case of NAs. The proposed procedure encompasses three main steps, depicted in figure 4 as a three-layer diagram.

1. *Input/pre-processing:* The initial step involves providing the input to the workflow, comprising a quantum circuit in which all platform-independent optimizations have already been performed.
2. *Computational capabilities:* This part involves the compilation process, which varies depending on the specific hardware capabilities, encompassing long-range interactions, multi-qubit gates, and shuttling as discussed in section 3.2.
3. *Figures of merit:* The final step requires using suitable figures of merit to evaluate the quality of the compilation, which may differ across various capabilities.

4.1.1. Input/pre-processing

The initial stage acts as the input layer in the proposed compilation overview. It outputs a fully optimized circuit where all hardware- and platform-independent optimizations have been applied. Because this phase is not tied to any particular platform, well-established optimization techniques and software initially designed for SC hardware can be employed here. The result is a sequence of gates, still including abstract gates that will be broken down in the subsequent steps based on the available hardware. Certain aspects of this process can also be pre-compiled beforehand, stored, and later retrieved from an intermediate representation [107, 108].

4.1.2. Computational capabilities

The subsequent layer includes hardware-dependent compilation steps, specifically *synthesis*, *mapping*, and *scheduling*. Unlike the previous optimizations, these steps depend on the available operations, such as the

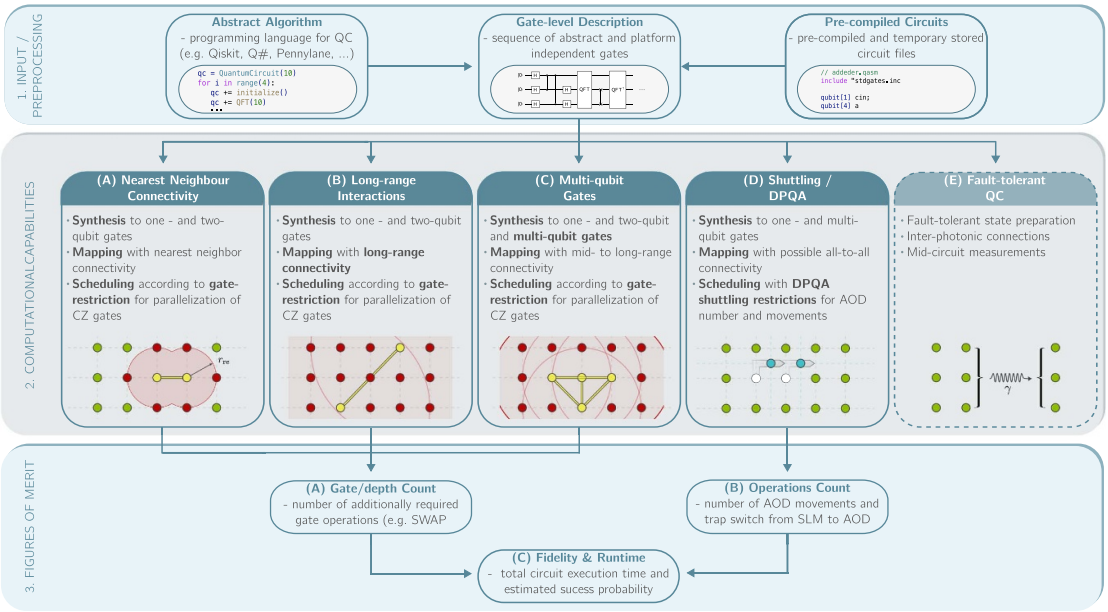

Figure 4. Compilation and evaluation process overview for the neutral atom quantum computing platform. 1. Input/Preprocessing: the platform-independent compilation processes lead to a circuit description that includes abstract and hardware-independent gates. 2. Computational capabilities: the different computational capabilities of the NAQC platform, as elaborated in section 3.2 with a short description of the constraints for the corresponding compilation subtasks of synthesis, mapping, and routing. Depending on the hardware setup, multiple capabilities, including all, can be considered. 3. Figures of merit: the compilation output is evaluated based on capability-specific proxy criteria, such as gate or shuttling operation counts. To achieve a more comprehensive comparison and evaluation of gate-based routing and shuttling, figures of merit, such as the final execution time and fidelity of the compilation result, can be computed.

basis gate set and how SWAP operations can be performed. Consequently, the compilation process may vary based on the set of available capabilities and the corresponding constraints, which directly correspond to the sections of section 3.2.

(A) *Nearest-neighbor connectivity:* The first option involves utilizing capabilities equivalent to those often considered for the SC platform. This includes employing single-qubit rotations and Rydberg blockade-based controlled phase gates, which differ from CX by just single-qubit operations. Together, they form a universal gate set for decomposing non-native gates during synthesis. Qubit connectivity follows nearest-neighbor connections, defining the connectivity graph G. Synthesis, mapping, and routing techniques applied in SC systems can be adapted with adjustable parameters such as gate time or fidelity. The additional constraint, *gate restriction* from section 3.2.2, must be considered on top as it is unique to the NAQC platform.

(B) *Long-range interactions:* One significant distinction of NAs is their ability to execute two-qubit phase gates not only between adjacent qubits but also between any two atoms where the blockade interaction reaches sufficient strength. This increases connectivity, reducing the need for additional SWAP operations during circuit execution. However, the larger restriction volume due to long-range interactions may lead to more sequential gate execution.

(C) *Multi-qubit gates:* Besides higher connectivity, the long-range Rydberg interaction enables the implementation of native multi-qubit gates, which is particularly beneficial for the synthesis task or algorithms that inherently consist of many multi-qubit gates, such as reversible classical logic circuits. However, involving more atoms also increases the restriction volume, potentially limiting simultaneous gate executions.

(D) *Shuttling/DPQA:* A fundamental advantage of the NAQC platform is the ability to physically move atoms, and hence qubits, rather than relying on virtual SWAP operations. High-fidelity shuttling is a promising alternative for gates between distant qubits. Additionally, a fully shuttling-based architecture (dynamically field-programmable quantum array—DPQA) might be possible, eliminating the need for virtual swaps altogether.

(E) *Fault-tolerant quantum computing:* NAs also offer additional capabilities relevant to future compilation tasks, such as fault-tolerant quantum computing. The details of compiling logical circuits are beyond

the scope of this work, but with the recent experimental progress, first work in this direction has been done, in particular discussions on the implementation of qLDPC codes Xu *et al* [100], surface codes [101], and generalized bicycle codes [102].

Theoretically, it is feasible to combine multiple, or even all, of the aforementioned capabilities. For instance, it is possible to harness both physical shuttling operations and virtual SWAP gates within the same computation. Alternatively, one could adopt the DPQA approach while organizing atoms into subgroups instead of pairs, thereby utilizing shuttling and the ability to implement multi-qubit gates within the atomic subgroups. Examples are given by Viszlai *et al* [101] studying possible surface code architectures to take advantage of such a setup, or mapping algorithms taking advantage of using both, gate and shuttling-based mapping techniques [36]. Nevertheless, there remains a diverse set of compilation tasks and open avenues for further investigation.

4.1.3. Figures of merit

The evaluation of a compilation output requires the use of appropriate performance measures or metrics. The choice of metrics varies depending on whether the aim is to compare different compilers, capabilities, or platforms. For instance, the common practice of counting additional swap gates as a measure in SC compilers lacks significance for a DPQA-based architecture that avoids swap gates altogether. Therefore, novel metrics may be required to measure the quality of a compilation output for NAs. The same holds for the figures of merit used during optimization in the form of cost functions. These figures of merit, suitable for NA-based hardware, are discussed in the following.

(A) *Gate count:* The predominant approach to evaluate the quality of a compilation is to quantify the total number of gates. Directing attention to the gates with the lowest fidelity often suffices, as they exert the most significant influence on the final output. In the context of synthesizing, notable metrics include the counts of CX gates for NISQ computing or T gates in the regime of fault-tolerant quantum computing. In the case of NAs, a similar figure of merit could be employed, where, depending on the hardware, the focus lies on the gates with the lowest fidelity or longest execution time.

In addition to the total number of gates, another critical aspect to consider is the number of gate layers, referred to as *circuit depth*, where each layer contains operations that can be executed simultaneously. This evaluation considers the scheduling of operations and serves as a proxy criterion for estimating the overall execution time. However, when dealing with NA, especially in cases involving nearest-neighbor interactions (2.A), it is imperative to account for the existence of gate restriction volumes, as elaborated in section 3.2.2, during the scheduling process. This will result in larger circuit depths in general.

(B) *Operation count:* Since shuttling does not introduce additional gates into the circuits, the previous gate-based measure is not applicable. An alternative approach would be to define a similar and straightforward operation count. A suitable candidate for this count could be the number of AOD-based MOVE or SWAP operations. For DPQA architectures, one can use the number of iterations of the three phases of loading, shuttling, and execution as a first-order evaluation measure.

(C) *Fidelity and runtime:* The diverse possibilities for implementing a quantum circuit on the NAQC platform render simple measures such as gate count less suitable, especially when multiple capabilities are leveraged. The most comprehensive measure for evaluating a quantum computation is the output fidelity and the total runtime. Unfortunately, computing the exact output fidelity is generally not feasible. To address this limitation, we considered the proxy of *approximate success probability* of equation (13), quantifying the likelihood of executing the circuit successfully without errors. It enables a comparative analysis of compilation outcomes for various circuits, facilitating the evaluation of a given compiler's performance. Additionally, this metric allows for comparing different capabilities, assisting in estimating the most promising approach for mapping, for example, whether employing nearest neighbor, long-range SWAP gates, or shuttling SWAP operations are most suitable.

4.2. Compilation parameters

All stages within the overview depicted in figure 4 are contingent upon hardware-specific parameters, such as the strength of the Rydberg blockade, the gate execution time for the mapping and -scheduling task, and the corresponding gate fidelities used to compute the final output fidelity. These parameters heavily depend on the chosen hardware and the overall experimental configuration, including the atom species and the protocol employed to implement specific gates. Nonetheless, the development of compilers necessitates the utilization of reasonable hardware parameters to generate practical results for hardware experts and other users. Hence, we present a list of estimated parameters in table 1 for two exemplary setups. These parameters align with the

Table 1. The hardware parameters pertain to the compilation steps of mapping and scheduling, alongside the subsequent evaluation measures shown in the overview of figure 4. These parameters are provided for two hypothetical experimental setups and are intended solely for a preliminary estimation, as they are dependent on the specific hardware utilized and are likely to change with future experimental improvements.

			Setup 1	Setup 2
General	Atom species		Strontium [56, 64, 109]	Rubidium [5, 9]
	Inter atomic distance	d	3 μm	3 μm
Gate durations t	Single-qubit		200 μs [109]	0.5 μs [110]
	CZ		0.1 μs [64]	0.2 μs [5]
	CCZ		$\sim$1 μs	$\sim$1 μs
	CCCZ		$\geqslant$1 μs	$\geqslant$1 μs
Gate fidelities $\mathcal{F}$	Single-qubit		>0.99 [109]	>0.999 [110]
	CZ		>0.99	>0.995 [5]
	CCZ		$\sim$0.95	>0.98 [5]
	CCCZ		$\sim$0.95	$\sim$0.95
Coherence times	Qubit decay	T_1	>1 s [56]	>100 s
	Dephasing	$T_2^{\S}$	>10 s [56]	>1.5 s [5]
Long-range	Interaction radius	r_{int}	2	2
	Restriction radius	r_{re}	4	4
Shuttling	Fidelity		1	1* [9]
	Shuttling speed	v_s	<0.025 μm μs^{-1} [109]	<0.55 μm μs^{-1}* [9]
	Trap swapping		20 μs [75]	20 μs [75]

According to figure 1(d) of Bluvstein *et al* [9] we assume perfect fidelity if the average shuttling speed is below the indicated threshold.

We consider T_2 times using additional dynamic decoupling techniques.

more abstract description of the physical capabilities, suitable for tool developers, and discussed in section 3.2. It is important to note that these parameters are not fixed and will inevitably evolve in the coming years with advancements in hardware and control systems. The underlying idea is that one can then reason on an updated set of the same or similar parameters but with the same general considerations, as all shown parameters can change with future hardware improvements. Therefore, the objective is to develop more adaptable compilers that can adjust to specific hardware configurations provided.

4.3. Discussion

Figure 4 provides a comprehensive overview of the whole hardware-dependent compilation process by bringing the summary boxes of section 3.2 in relation to each other and to possible figures of merit for evaluation. Nevertheless, it should not be viewed as an inflexible framework dictating the compilation process. Instead, it should be perceived as a graphical representation delineating the broad spectrum of compilation possibilities of the NAQC platform. This complexity arises from the potential selection or amalgamation of multiple capabilities to achieve improved outcomes. In addition to the increased number of compilation options, the overview underscores the need for additional or more intricate metrics and figures of merit, depending on the available capabilities. Depending on the available hardware, a compiler can take multiple paths along the three layers, leveraging different capabilities. As a tool developer, one must check which paths the compiler should cover and, therefore, which figures of merit are suitable for optimization. In the following section 5, we will consider already available examples of compilers and discuss which path in figure 4 has been chosen. Afterward, in section 6, we will perform multiple selected case studies to compare error rates along different paths and discuss appropriate figures of merit, depending on the given hardware parameters.

5. Related work and software

Based on the compilation overview of figure 4, we want to give a concise summary of already available compilation software for NAs. For this aim, we categorized the compilers based on the capability they focus on most, including long-range, multi-qubit gates, and shuttling. Additionally, we also discuss the usage of SC compilers for the NAQC platform, since we will employ such an approach for some of the case studies in section 6. We briefly discuss each compiler's functionality and the corresponding methods used. All compilers and the respective supported capabilities are summarized in table 2. This section aims to review

Table 2. Overview of Presently Available Compilers for the NA Platform. The symbol ☑ indicates the main functionality of the software and implies full support, while ☑ refers to partial support only or reliance on external software. The setup differentiates between compilers that assume individual addressability of entangling gates and the DPQA setup discussed in section 3.2.4.

Capabilities	SC [13, 18, 19]	Baker *et al* [33]	Q-Tetris [28]	Geyser [29]	Brandhofer *et al* [31]	OLSQ-DPQA [111]	Nottingham *et al* [32]	Schmid *et al* [36]	FPQA-C [35]	Q-Pilot [112]	Tan *et al* [34]
Long-range											
Synthesis	☑			☑*		☑*	☑*	☑*	☑*	☑*	
Mapping & Routing	☑	☑	☑	☑§	☑†		☑‡	☑	☑§		
Scheduling		☑	☑	☑			☑	☑	☑		
Multi-qubit											
Synthesis				☑**							
Mapping & Routing	☑	☑	☑	☑				☑			
Scheduling		☑	☑	☑				☑			
Shuttling											
Mapping					☑§§	☑	☑	☑	☑	☑	
Scheduling					☑	☑	☑	☑	☑	☑	
Single-qubit											
Synthesis							☑				☑
Scheduling	☑	☑	☑	☑	☑		☑	☑	☑	☑	☑
Setup	indiv.	indiv.	indiv.	indiv.	indiv.	DPQA	indiv.	indiv.	DPQA	DPQA	—
General											
Technique/ Algorithms	SMT, A*, heuristics,....	look-ahead heuristic	heuristic, MCTS	heuristic	SMT	SMT, heuristic	heuristic	heuristic	MAX k-Cut, heuristic	heuristic	SMT
Open source availability	Github, PiPI,...	Github [113]	Github [114]	Zenodo [115]	—	Github [116]	—	Github [117]	—	—	—
Language	Python, C++, Rust,....	Python	C++	Python		Python		C++			
General input (e.g. QASM)	☑	☑	☑	☑		☑		☑			

conversion to native gate set (CZ-gates).
uses mapping from [13].
uses mapping from [21].
uses mapping from [15].
Composition of Toffoli gates.
only 1D displacement of grid row/columns.

currently available compilers and aids interested tool developers in identifying subproblems that still lack a corresponding software solution.

5.1. Compilers for superconducting platform

Before discussing compilers specific to the NAQC platform, we first want to briefly discuss the idea of reusing already available compilers. While general compilation for NAQC has to consider a more extensive range of capabilities and constraints, it is still possible to leverage e.g. currently available SC compilers with additional pre- and post-processing steps to get compilation outputs valid for NAQC. In this way, the full potential of NA-specific capabilities cannot be fully exploited. However, we still want to mention this possibility here, as we used similar techniques for the evaluations in section 6. Furthermore, SC compilation algorithms may provide a suitable basis for generalization and adaptation to the NAQC platform.

The SC platform has access to a broad range of advanced software tools [13, 14, 18–22] that greatly facilitate the compilation process. However, when applied to the NAQC platform, these tools are only partially suitable for the synthesis, mapping, and scheduling steps.

- *Synthesis*: As the SC platform lacks straightforward support for multi-qubit gates, the corresponding synthesis steps usually decompose to one- and two-qubit gates, omitting capability 2.C from figure 4.
- *Mapping*: SC relies on gate-based virtual swapping to establish the required connectivity, excluding 2.D from the process. However, the mapping problem can be adapted for NAs by defining the coupling graph **C** based on the interaction radius r_{int} as shown in equation (6).
- *Scheduling*: Additional scheduling constraints of the restriction volume according to equation (7) require an extra scheduling post-processing step for the NAQC platform.

Evaluation proxies commonly used in these tools involve gate count and circuit depth. This leads to the following path in figure 4: 1 → 2.A/2.B → 3.A.

5.2. Long-range compiler

When directly compared to other QC platforms, one of the constraints of the NAQC platform is the occurrence of restriction volumes whenever an atom employs the Rydberg state to execute a two- or multi-qubit gate. This significantly impacts the scheduling task, as such gates that restrict each other must be executed sequentially rather than in parallel, increasing the total execution time. Given that quantum information can only be stored for a limited time without decoherence, this directly affects the output fidelity, which can be estimated according to equation (13).

1. The initial solution to address this problem was proposed by Baker *et al* [33], with an openly available Python package [113]. Their approach assumes that the synthesis task has already been performed and concentrates on the mapping part and potential strategies to mitigate atom loss.

- *Mapping*: Utilizing a look-ahead scheme to select the shortest SWAP path with minimal disruption for future interactions.
- *Scheduling*: Execute swap gates in parallel when they do not impose restrictions on other operations.

Baker *et al* [33] primarily investigated the trade-off between long-range interactions and the effect of the simultaneously increasing restriction volume.

Remark. It should be noted that Baker *et al* consider a slightly different definition of the restriction volume than the one proposed in our work. Additionally, they assume the restriction to be variable, depending on the inter-qubit distance of the gate and, therefore, possibly varying from gate to gate.

The Mapper also supports multi-qubit gates, enabling the utilization of the full gate-based mapping capabilities of the NAQC platform, represented as 1 → 2.A-C → 3.A in figure 4.

2. Addressing the same problem formulation, Li *et al* [28] presented a C++ based solution named *Q-Tetris*, inspired by the resemblance of the problem to the renowned block puzzle game. In contrast to Baker *et al*, they consider distinct execution times for single, multi-qubit, and SWAP gates.

- *Mapping & scheduling*: Employ a greedy heuristic algorithm in conjunction with a Monte Carlo tree search (MCTS) approach to strategically insert necessary SWAP gates and simultaneously minimize the overall circuit execution time.

By arranging the gates in this time-aware manner, they claim that their method produces more time-efficient circuits compared to Baker *et al* Hence, they do not consider the gate count but the total execution time of

the circuit. Nevertheless, their approach supports the same capabilities as Baker *et al*, leading to the compilation path 1 → 2.A-C → 3.*A* → 3.*C* in figure 4.

5.3. Multi-qubit compiler

Although both previous compilers only support the mapping and scheduling tasks, as of our current knowledge, there is no available software capable of performing multi-qubit-aware synthesis for general quantum algorithms. For classical reversible logic, tools [118, 119] exist that create circuits containing Toffoli and higher controlled NOT gates.

1. However, Patel *et al* [29] and the corresponding open-source Python software *Geyser* [115] take a different approach. Instead of decomposing a large unitary into a basis gate set containing multi-qubit gates, they compose blocks of Toffoli gates from a set of two-qubit gates.

- *Synthesis*: The approach involves identifying gate blocks containing three qubits and seeking an equivalent gate sequence built from Toffoli gates and single-qubit gates. This is achieved by minimizing the Hilbert-Schmidt distance between the original gates and the substitute gates.
- *Mapping*: The SC compiler Qiskit [13] is used, considering that the constructed Toffoli blocks do not restrict each other.

For evaluation, they do not consider gate count but rather laser pulse count, which can be seen as an estimation for the total execution time of the circuit. In terms of the overview in figure 4, this leads to a similar path as the previous two compilers, 1 → 2.A-C → 3.A → 3.C, but with a focus on the previously unexplored synthesis task, allowing the possibility of leveraging both approaches.

5.4. Shuttling compiler

All previous compilers have solely considered the virtual gate-based SWAP operation for the mapping task. However, the Rydberg platform offers the additional capability of physically rearranging atoms to achieve an equivalent SWAP or a simple MOVE operation.

1. The pioneering attempt to harness this extra degree of freedom was made by Brandhofer *et al* [31]. Their approach does not encompass general shuttling operations but focuses solely on one-dimensional displacements of atom rows. This choice is justified because atoms trapped in rows or columns using AOD traps can be efficiently shuttled in parallel.

- *Mapping*: The model involves considering SWAP gate insertion (similar to SC [21, 120]) and potential one-dimensional displacements. The authors employ Satisfiability Modulo Theories (SMT) [121] solvers to find an optimal solution.

They compute the final circuit fidelity to evaluate the results and compare gate-based swapping with displacement-based swapping. Consequently, the compiler path in figure 4 can be described as 1 → 2.A/B → 3.A → 3.C and 1 → 2.D → 3.B → 3.C, covering a significant portion of the NA capabilities.

Rather than considering only one-dimensional displacements, Tan *et al* [111] focus on a shuttling-only DPQA architecture as discussed in section 3.2.4. The corresponding open-source Python package [116] enables users to explore DPQA movements and offers helpful visual animations of these operations.

- *Mapping & scheduling*: They encode the possible DPQA movements as an SMT problem and solve it to optimize the number of shuttling operation cycles. Additionally, they propose a heuristic approach based on the exact solver with better runtime scaling.

In the overview figure 4, this compiler corresponds to the path 1 → 2.D → 3.B, but this time it takes advantage of the full two-dimensional shuttling capability.

2. Recently, Nottingham *et al* [32] presented a novel compiler that addresses the crucial synthesis step for the NAQC platform, considering the global addressing of the qubits. Additionally, they propose a unique mapping approach for a shuttling-only architecture, with less stringent constraints compared to Tan *et al* [111].

- *Synthesis*: The authors propose two distinct decomposition approaches to address the challenge posed by single-qubit rotations that are only available as global beams. They leverage Qiskit's internal decomposition to single-qubit and CZ gates and apply their own optimization post-processing.
- *Scheduling*: Accounting for global beams in single-qubit gates, the gates are organized into groups of parallelizable single-qubit gates and parallelizable CZ gates. CZ gates that restrict each other are executed sequentially.

- *Mapping*: By relaxing the shuttling constraints from equations (10) and (11), they allow different AOD rows/columns to cross. This enables the definition of a *movement graph* that indicates all possible shuttling moves analogous to the connectivity graph for gate-based interactions. Mapping is then performed on this denser graph by selecting shuttling movements that bring the gate qubits closer to each other.

Nottingham *et al* [32] marks the first step toward synthesis for NAs, addressing global single-qubit rotations and CZ gates. They subsequently evaluate both virtual swapping and shuttling-based swapping in a DPQA-like problem scenario. Their evaluation accounts for the influence of a gate-based error model and decoherence errors. Regarding the compilation overview, this approach can be described as $1 \rightarrow 2.A/B/D \rightarrow 3.A/B \rightarrow 3.C$, encompassing all NA-specific capabilities except for native multi-qubit gate support.

3. Supporting all capabilities of figure 4 was done by Schmid *et al* [36], proposing a hybrid compilation scheme where a heuristic decides the most suitable mapping capability for each gate.

- *Mapping*: A SABRE-based [16] heuristic is used to compute the necessary SWAP gates. The duration and fidelity of these are compared to AOD-based shuttling operations, choosing the more favorable option.
- *Scheduling*: Assumes single addressability and taking into account multi-qubit gate restrictions. Shuttling operations are converted to the respective AOD movements and scheduled according to the constraints in equations (10) and (11).

This approach allows to take advantage of the full spectrum of capabilities and covers all paths in figure 4 except the consideration of fault-tolerant quantum computing.

4. A complementary approach, also employing a combination of shuttling operations and SWAP gates is FPQA-C [35]. Based on the FPQA setup, SWAP gates are used to connect atoms, which cannot be brought close together due to the AOD constraints.

- *Mapping & Scheduling:* First, the qubits are assigned to different atom arrays with one trapped in static SLM traps and the others controlled by a separate AOD each. This is done using a MAX k-cut heuristic to minimize the intra-array entangling gates, which require additional SWAP gates to connect. Inter-array gates are performed by rearranging the AOD coordinates according to equations (10) and (11).

In this sense, FPQA-C represents a hybrid mapper for the DPQA setup, currently taking into account two-qubit gates only, resulting in $1 \rightarrow 2.A/B/D \rightarrow 3.A/B \rightarrow 3.C$.

5. Wang *et al* [112] propose the use of so-called *flying ancillas* within the FPQA setup, refered to as Q-Pilot. Instead of bringing the gate qubits close to each other, a third ancilla qubit is shuttled between them to establish the entanglement. In this way, trapping only the ancilla qubits in movable AODs eliminates the switching between AOD and SLM traps.

- *Mapping & Scheduling:* The movements of the ancilla qubits are computed based on a heuristic taking into account equations (10) and (11), with a specialized version for QAOA circuits.

While the work concentrates on mapping everything using flying ancillas, this can be considered as an alternative or complementary mapping approach using shuttling. The respective path is $1 \rightarrow 2.D \rightarrow 3./B \rightarrow 3.C$.

In addition to the capability-specific compilers, Tan *et al* [34] proposed depth-optimal addressing of single qubit gates on a two-dimensional grid using SMT solvers. As this represents a scheduling-only problem, it can be considered in addition to any of the four capability paths, assuming the hardware supports this type of single qubit gate execution.

5.5. Overview and discussion

A summarized overview of all compilers and their respective functionalities are provided in table 2, categorized based on the supported computational capabilities of section 3.2. These correspond directly to the paths in the second layer of figure 4 and the entailed figures of merit. Note that this should be considered as an abstract overview at the current point in time. The assignment to the different compiler functionality may vary depending on the exact definitions of the compilation steps and can also change in the future with the ongoing development of the tools.

However, this overview also reveals that numerous open problems still need to be addressed. Notably, one significant challenge is the synthesis step, which involves decomposing general quantum algorithms into one- and two-qubit gates and the full set of supported multi-qubit gates. Additionally, there is a need to

explore the combination of multiple capabilities as done by for example by FPQA-C [35] and Schmid *et al* [36]. In this regard, in section 6.4, we perform case studies and error analysis to illustrate possible metrics to decide between multiple capabilities to perform the same operation.

In summary, while progress has been made in addressing NA-specific compilation challenges, further advancements are needed to fully exploit the platform's capabilities and simplify its utilization for both hardware experts and computer scientists. The aim of the 'big picture' view proposed and discussed in this section is to consider the full range of hardware capabilities instead of studying them independently of each other. This results in compilers that take full advantage of the NAQC platform, necessary to compete with other hardware platforms. For this aim, it is essential to evaluate promising capabilities for their usefulness and study hardware-dependent error sources to find valuable figures of merit that can be employed as cost functions during optimization. This is done and discussed in the following section 6, with complementary evaluations of the NA capabilities discussed in other recent work [122, 123].

6. Selected case studies

We now proceed to study the platform-specific capabilities of NAs in the compilation context through selected case studies. More concretely, we evaluate when and how much the platform-specific features and limitations impact the compilation process. This is done by employing existing compilers from section 5 and analyzing their outcome based on the approximate success probability metric outlined in section 3.2.6. The aim is to provide estimations on error sources considering current and imminent hardware configurations.

These considerations offer insights for compiler developers, aiding in identifying predominant error sources and refining optimal compilation software. This section does not aim at providing a final decision on the most suitable capability but to exemplarily discuss potential techniques to compare among them. In particular, it provides information on suitable figures of merit, depending on a given hardware setup. For hardware experts on the other side, the discussion provides a forward-looking perspective for devising future hardware designs, concentrating on the most promising capabilities within the NAQC platform. The concepts discussed in this section can be generalized and used also for example in the context of compiling for DPQA setups.

First, a brief discussion on the error analysis based on the summary box of section 3.2.6 is given, focusing on the mapping and scheduling part only. The aim is to get an easy-to-compute metric to evaluate and compare different mapping and scheduling approaches. With this study, we can get insights about interesting figures of merit by comparing different hardware setups. In particular, the two most commonly used figures of merit are the questions of whether compilation passes should be optimized for gate count or circuit depth.

As a second step, we apply this analysis to numerically analyze and discuss the NA-specific capabilities, namely long-range, multi-qubit gates and shuttling. In this case, we aim to identify the most promising capability, depending on some given hardware parameters. This includes open questions such as the required gate fidelity or shuttling velocity to achieve an advantage for one of the two possible mapping capabilities. Both considerations are of importance for both tool developers and hardware experts. Software tool developers get insights into how hardware parameters can affect the compilation process and change the choice of figures of merit. Concurrently, the latter can estimate the potential of given capabilities, aiding them in devising forthcoming hardware configurations.

6.1. Error metrics and code availability

For the evaluations, we adopt the hardware parameters detailed in table 1. Following the compilation process outlined in section 4.1, we examine the three distinctive capabilities of long-range interactions, multi-qubit gates, and atom shuttling. Depending on the scenario, we will vary hardware parameters, compilation constraints, and the employed compilation software to study their impact on the error metrics discussed in section 3.2.6.

6.1.1. Fidelity estimation

Due to the absence of NA-specific synthesis software, our focus will be on the mapping and scheduling stages in the subsequent sections. We compute the final output fidelity, as introduced in equation (13), as our performance metric. As we will focus on the mapping and scheduling steps, the considered circuits are all fully decomposed. Therefore, the gate errors of different mapped circuits differ only on the added SWAP gates. When comparing different mapping approaches, it is, therefore, sufficient to focus on the fidelity reduction due to SWAP gates and idle errors instead of computing the full approximate success probability P as discussed in section 3.2.6.

For gate-based swapping, this leads to the following commutative fidelities regarding mapping and scheduling:

$$\mathcal{F}_{\text{SWAP}} = \mathcal{F}_{\text{idle}} \cdot \mathcal{F}_{\text{mapping}} = \exp\left(-\frac{t_{\text{idle}}}{T_{\text{eff}}}\right) (\mathcal{F}_{\text{CX}})^{3 \cdot N_{\text{SWAPS}}} . \tag{16}$$

Here, t_{idle} corresponds to the idle time of the register according to equation (15), and N_{SWAPS} represents the total count of introduced SWAP gates, each executed through three CX gates. The value of N_{SWAPS} can be extracted directly from the compilation results. Furthermore, we assume CX to be realizable by a combination of one single-qubit gate and a CZ gate and, therefore, add the corresponding gate durations and multiply the fidelities to get the parameters for the CX gate. For the computation of t_{idle}, we used Python-based post-processing scripts to correctly map and schedule the circuit gates on a given hardware. A visualization of the scheduling is available together with the scripts at Zenodo [124].

For shuttling-based swapping, on the other hand, we assume a process fidelity of 1 (table 1) for the shuttling procedure, making decoherence during idle the sole source of error.

$$\mathcal{F}_{\text{sh}} = \exp\left(-\frac{t_{\text{idle}}^{\text{sh}}}{T_{\text{eff}}}\right) . \tag{17}$$

Here, $t_{\text{idle}}^{\text{sh}}$ represents the idle time of the register, incorporating the time necessary for performing the shuttling operations.

6.1.2. Software and code availability

All the compilers employed in the subsequent analysis are accessible as open-source software. These tools have been installed and used in adherence to their respective documentation. The evaluation and interpretation of data, encompassing tasks like scheduling and error computation, have been realized in Python. The complete set of scripts employed to generate and visualize the ensuing outcomes, along with the analysis data, can be found on Zenodo [124]. This collection also encompasses a list of utilized software packages with their corresponding version numbers.

The considered quantum circuits are taken from MQT Bench [125], a benchmarking suite containing commonly used quantum algorithms. In particular, we selected the following from the collection of benchmarks.

- **dj:** Deutsch-Jozsa algorithm.
- **ghz:** Preparation circuit of the Greenberger– Horne–Zeilinger state.
- **graphstate:** Circuit corresponding to a 2-regular random graph with edges representing two-qubit gates.
- **qft:** Quantum Fourier Transform.
- **twolocalrandom/two-local:** The two-local Variational Quantum Eigensolver ansatz with random parameters.
- **wstate:** Preparation circuit of the entangling W-state.

6.2. Long-range interactions

First, we want to study the long-range capability and the question regarding the trade-off between high connectivity and the simultaneous gate restriction. In contrast to the considerations already done in Baker *et al* [33], we focus on the question of whether the compilation pass should minimize SWAP gates or gate restriction, depending on different fidelities and coherence times.

Performing mapping with long-range interactions implies gate-based mapping incorporating a fidelity $\mathcal{F}_{\text{mapping}}$ and $\mathcal{F}_{\text{idle}}$, as given in equation (16). The former arises due to the introduction of extra SWAP gates, while the latter emerges from gate durations and restrictions, resulting in idling qubits. In the following, we aim to investigate the circumstances under which each error contribution supersedes the other, highlighting potential optimization avenues on both the hardware and software fronts. Furthermore, we investigate, if familiar figures of merit like SWAP gate count or circuit depth are applicable for NAs. For this aim, first, we will study different hardware configurations by varying parameters r_{int} and r_{re}, followed by utilizing two compilers optimizing for different figures of merit.

6.2.1. Interaction and restriction radius

A larger interaction radius r_{int} yields increased connectivity, resulting in a reduction of SWAP gates and subsequently improving $\mathcal{F}_{\text{mapping}}$. However, the stronger influence of long-range Rydberg interactions also imposes limitations on a greater number of nearby atoms. As discussed in section 3.2.2 this impedes gate parallelization, amplifying qubit idling, and thereby reducing $\mathcal{F}_{\text{idle}}$. Depending on hardware parameters,

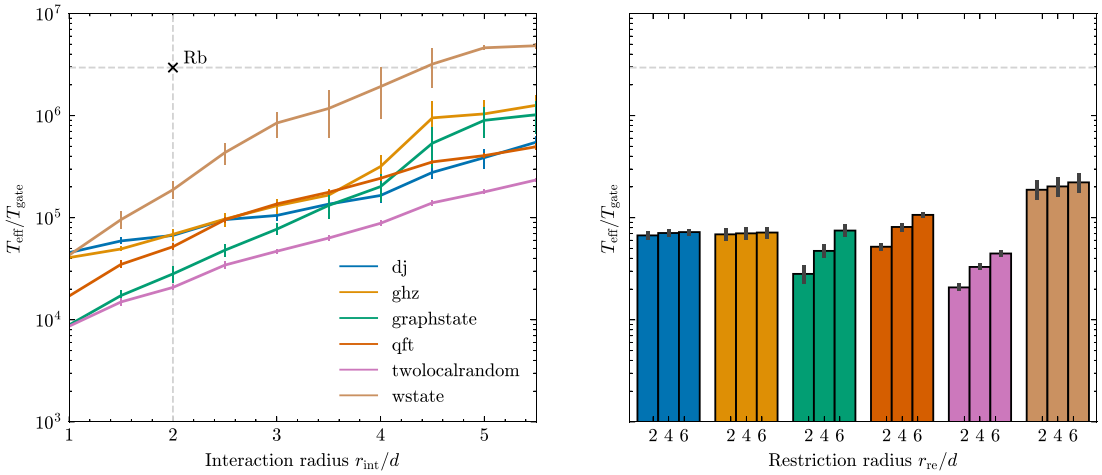

Figure 5. Long-range interaction analysis: in the left graph, we computed the fidelity reduction based on SWAP gate insertion and qubit idling for diverse circuits with a dimension of $n = 120$ and varying interaction radii r_{int}. The displayed curve represents the effective coherence time required to equate $\mathcal{F}_{\text{idle}}$ and $\mathcal{F}_{\text{mapping}}$. This signifies that, given a circuit, when T_{eff} surpasses (falls below) the line, the dominant fidelity reduction is caused by SWAP gates (decoherence). Larger r_{int} diminishes the SWAP error due to augmented connectivity, necessitating larger coherence times to balance the influence of both errors. Owing to the prolonged coherence times of the Rubidium hardware, SWAP gates stand as the prevailing source of fidelity reduction across all circuits. The right graph follows a similar procedure, modifying the restriction factor r_{re} while maintaining a fixed interaction radius of $r_{\text{int}} = 2d$, showing different behavior depending on the structure of the circuit. The error bars correspond to the standard deviation averaged over 10 iterations.

such as T_{eff}, one of these two factors becomes the predominant source of error. We employ the Qiskit compiler [13] to map circuits comprising $n = 120$ qubits onto a square grid, followed by post-processed scheduling. We evaluate the effects by computing the coherence time per gate duration $T_{\text{eff}}/T_{\text{gate}}$, where $\mathcal{F}_{\text{idle}}$ and $\mathcal{F}_{\text{mapping}}$ are equal. Placing hardware parameter sets in the same graph thus provides a direct guideline for future improvements. First, we varied r_{int} with a fixed restriction factor $k = 1$, while afterwards, we modified the restriction radius $r_{\text{re}} = kr_{\text{int}}$ for $k = 1, 2, 3$, maintaining a constant $r_{\text{int}} = 2d$. Each configuration is averaged over 10 runs to account for the stochastic character of the SABRE compiler.

The results are shown in figure 5. For all the algorithms considered, an increment in r_{int} leads to a corresponding rise in T_{eff} needed to achieve a balance between gate and decoherence errors. For a given circuit, data points situated above the depicted line signify a higher T_{eff}, indicating that the primary source of reduced fidelity is SWAP gate insertion. Conversely, points below the line indicate a larger decoherence error. For the Rubidium hardware configuration as detailed in table 1, the dominance of SWAP gate errors holds true for nearly all scenarios. Except for instances of large interaction radii, such as $r_{\text{int}} \geqslant 4.5$ for the W-state preparation, decoherence errors begin to supersede SWAP gate errors.

On the right side of the graph, we observe how this point of error equivalence is influenced by altering the restriction radius. Evidently, algorithms featuring a more sequential gate arrangement, like the GHZ state circuit, are less impacted by greater restrictions. In contrast, circuits such as the Graphstate circuit or the Two-local ansatz, characterized by numerous parallel gate executions, experience more pronounced effects from an increased restriction radius. This is due to the more sequential execution of the gates, as a parallel execution is impossible due to gate restrictions. Subsequently, the total idle time increases and, therefore, also the decoherence error.

After varying hardware parameters and studying the influence on the output fidelity, we now want to use two compilers in the following, each optimizing for different figures of merit.

6.2.2. Compiler comparison

With all hardware parameters held constant, the exploration now extends to using two different compilers, each optimized for a distinct figure of merit. In particular, we utilize the Qiskit internal SABRE heuristic [16], which aims to minimize N_{SWAPS}, and the Q-Tetris heuristic [114] elaborated upon in section 5.2. All runs are repeated four times and averaged to account for the stochastic character of SABRE. Q-Tetris orchestrates gate arrangements to amplify parallelism, hence reducing t_{idle}. This trend is evident in table 3, demonstrating that Q-Tetris, on average, yields a 20%–30% reduction in idle time, albeit at the expense of requiring a notably larger number of SWAP gates. This results in up to double the SWAP gate count when compared to Qiskit. This divergence becomes particularly pronounced for larger r_{int} values. Consequently, Qiskit prioritizes minimizing N_{SWAPS} and, therefore, optimizing $\mathcal{F}_{\text{mapping}}$, while Q-Tetris

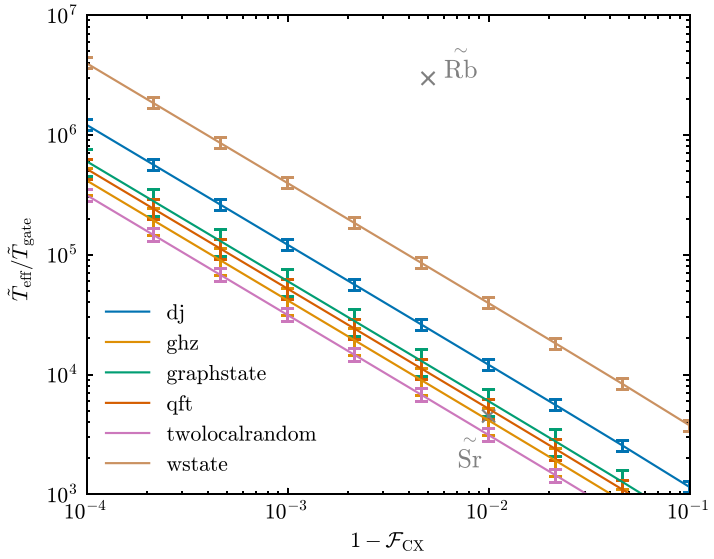

Figure 6. Comparison of compiler mapping fidelity: by utilizing the values from table 3, one can derive the gate-based fidelity reduction $\mathcal{F}_{\text{SWAP}}$ for the outcomes of the compilers. Each circuit displays a line depicting the point at which the fidelity of both compiler outputs matches exactly while varying the CX fidelity $\mathcal{F}_{\text{CX}}$. Given that Qiskit's compilation outcomes involve fewer SWAP gates but result in longer idle times, instances lying above the line (and therefore larger $\tilde{T}_{\text{eff}}$) should opt for the Qiskit compiler. In contrast, those lying below signify lower coherence times, thus favoring Tetris to curtail qubit idling. The error bars correspond to the standard deviation over four runs of the SABRE mapping.

focuses on achieving optimal t_{idle}. Variations in CX fidelity and coherence times favor one of the two fidelities thereby selecting one of the two compilers can be favorable.

It is important to note that for this evaluation, the utilization of Q-Tetris adheres to the gate durations found in Li *et al* [28]. Specifically, it assumes that CX (SWAP) gates require 2× (6×) the execution time of a single-qubit gate. The definition of qubit restriction aligns with that presented in Baker *et al* [33], which differs from the definition given section 3.2.2. Consequently, we solely consider the single-qubit gate time from table 1 and compute the CX (SWAP) gate duration as the corresponding multiple. Therefore, the hardware parameters are indicated with a tilde (∼) to emphasize that they do not directly correspond to the actual hardware parameters.

As our evaluation is based on rudimentary proxy criteria for the errors in question, it is imperative to treat these findings as approximations or rough estimates.

In figure 6, the computation of the effective coherence time $\tilde{T}_{\text{eff}}$, where both error components balance, is showcased. Large coherence times increase idle fidelity $\mathcal{F}_{\text{idle}}$, rendering the Qiskit compiler more favorable. Hence, for data points situated above (below) the line, the Qiskit (Q-Tetris) compiler should be chosen, respectively.

When contrasting the results against the two hardware configurations listed in table 1, it becomes evident that, for the Strontium scenario, both compilers yield comparable outcomes across most algorithms, barring the W-state preparation and the Deutsch-Jozsa algorithm, where the Q-Tetris compiler is more suited. This becomes clear, in particular for the W-state, considering again table 2. Q-Tetris is able to reduce the idle time of about 60 ms while adding only 50 additional SWAP gates compared to Qiskit. For other circuits, the cost of reducing the idle time is at a larger increase in the SWAP gate number. For these considerations, one must take into account that the fidelity difference between the two compilation outputs is based on the absolute difference in t_{idle} and N_{SWAPS}, while in table 2 only the relative difference is shown. For the Rubidium hardware, on the other hand, the Qiskit compiler remains advantageous for all cases due to the extended coherence times in relation to the short gate duration.

6.2.3. Discussion—long-range

Examining the process of gate-based swapping reveals the emergence of two significant sources of fidelity reduction: the insertion of SWAP gates and the occurrence of decoherence during qubit idle periods. The numerical error analysis provides insights, benefiting both hardware experts and tool developers in their efforts to improve the final fidelity of the results.

Table 3. Compiler comparative analysis: a comparison is drawn between the Qiskit compiler [13] and the Tetris compiler [114], focusing on the resultant idle time t_{idle} and the count of inserted SWAP gates N_{SWAPS}. While Qiskit prioritizes the minimization of N_{SWAPS}, Tetris aims at enhancing parallelism, leading to a reduction of t_{idle}. The final column showcases the relative variance of Tetris in comparison to Qiskit, elucidating that each compiler performs best in its respective domain. For Qiskit the numbers are averaged over four runs.

		t_{idle} (μs)			N_{SWAP}		
	r_{int}	Qiskit	Tetris	%	Qiskit	Tetris	%
dj	1	187 352	119 887	−36.0	314	500	+59.0
	2	113 363	66 703	−41.2	152	372	+145.1
	3	72 582	49 099	−32.4	84	209	+150.0
ghz	1	97 847	74 545	−23.8	181	368	+102.9
	2	59 990	52 147	−13.1	81	227	+179.6
	3	48 855	37 669	−22.9	45	121	+167.7
graph	1	31 893	17 340	−45.6	434	515	+18.6
	2	23 035	16 926	−26.5	182	357	+95.7
	3	20 683	14 412	−30.3	96	248	+158.3
qft	1	1108 553	813 101	−26.7	6117	8010	+31.0
	2	1177 610	900 108	−23.6	2849	5576	+95.7
	3	1470 671	979 202	−33.4	1479	3481	+135.3
two-local	1	2172 426	1239 620	−42.9	29 989	39 882	+33.0
	2	2118 934	1462 344	−31.0	14 252	24 102	+69.1
	3	2750 812	2035 308	−26.0	7446	18 188	+144.3
wstate	1	153 827	94 693	−38.4	360	410	+13.8
	2	106 314	72 741	−31.6	129	245	+89.9
	3	90 755	68 965	−24.0	68	164	+139.8

Hardware experts have the opportunity to engage in similar considerations as those demonstrated in figure 5, allowing them to analyze how hardware parameters influence the execution of specific circuits. Moreover, it becomes feasible to estimate the potential enhancements that hardware modifications could bring about. For instance, in figure 5, further increasing T_{eff} would result in only partial error improvements as the SWAP error due to small r_{int} overshadows the improved decoherence errors.

Compiler developers, on the other hand, will find this discourse to be of significant relevance. While the prevailing optimization objective is often considered the reduction of inserted SWAP gates, figure 6 illustrates that the key figure of merit to optimize for can substantially vary based on hardware parameters such as T_{eff} and $\mathcal{F}_{\text{CX}}$.

This underscores the necessity for a more flexible and adaptable compiler strategy that can adjust its optimization cost function following the specific hardware configuration.

Lastly, the observation that different circuits are influenced to varying degree by the constraints imposed by gate restrictions provides advanced compilation strategies with the opportunity to choose suitable hardware configurations for each type of circuit. Specifically, sequentially structured circuits, such as the Deutsch-Jozsa algorithm, could be scheduled for hardware setups characterized by higher r_{re}, given that such algorithms are less sensitive to gate restriction.

6.3. Multi-qubit gates

The primary focus on using native multi-qubit gates is during the synthesis step. This involves exploring whether simpler decompositions can be achieved using a broader set of native gates, including advanced gates such as the Toffoli gate. Unfortunately, we observe currently a lack of software tools available to address this issue, and the development of such algorithms for the NAQC platform remains a mostly unresolved challenge.

Another approach is to look at circuits that already contain multi-qubit gates, for example, for reversible classical logic, and compare the native execution of the gate with the corresponding decomposition. While multiple approaches exist for decompositions [126, 127], we want to focus on direct decomposition without additional auxiliary qubits. In this case, the multi-quit gate can be substituted one-to-one with the corresponding decomposed set of gates. As a result, the native execution is favorable if it has a smaller error in terms of fidelity and decoherence compared to the decomposition.

To compare the native multi-qubit gates with their decomposition we employed the Qiskit transpiler function to get a decomposition of the CCZ and the CCCZ gates in terms of single-qubit gates and CZ gates.

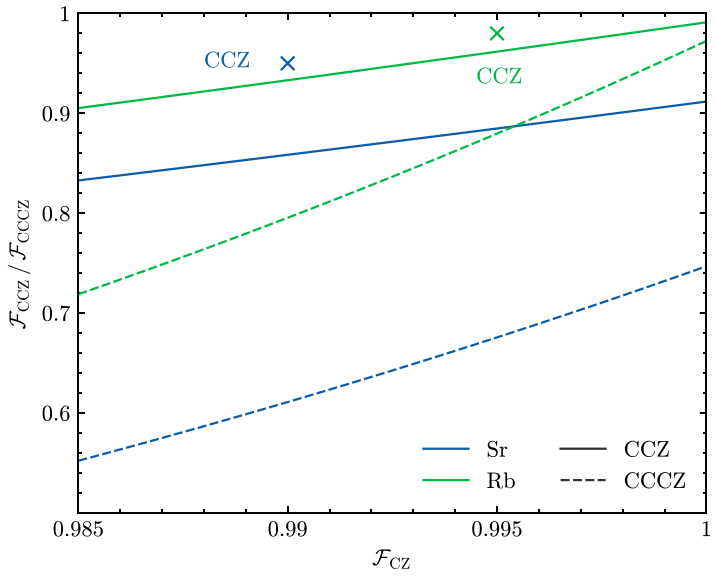

Figure 7. Required multi-qubit gate fidelity: the lines indicate the multi-qubit gate fidelity required for the CCZ (solid) and CCCZ (dashed) gate, to be favorable to the approximate success probability (section 3.2.6) of the decomposition. The fidelity is computed for varying CZ fidelities. The respective decompositions have been found employing the Qiskit transpile function. This threshold is depicted for both hardware setups, Strontium (blue) and Rubidium (green) with the parameters taken from table 1. For the CCZ gate, indicated by the crosses, the native implementation is preferable compared to the decomposition. The general lower success probability of the Strontium setup is due to worse single-qubit fidelity and longer gate durations.

The decompositions are constructed of 9 single-qubit, and 6 CZ gates for the CCZ gate, and 28 single-qubit, and 20 CZ gates for the CCCZ gate. Subsequently, we computed the approximate success probability P from section 3.2.6 for the native gate and the decompositions respectively. Figure 7 depicts the multi-qubit fidelity necessary depending on the CZ fidelity $\mathcal{F}_{\mathrm{CZ}}$. Additionally, the parameters from table 1 are indicated for the CCZ gate, showing that for both hardware setups, the native implementation is favorable compared to the found decomposition. This is in particular interesting for the Rubidium setup, where the CCZ gate with the corresponding fidelity has been demonstrated in experimets [5]. The worse performance of the Strontium setup for the same $\mathcal{F}_{\mathrm{CZ}}$, is mainly due to the single-qubit gates with lower fidelity and longer gate durations.

From a compilation point, this step is trivial, as the decomposition error can be computed beforehand according to equation (14), and depending on the outcome, the substitution can be performed or not. According to figure 7, the native gate implementation would be favorable if its fidelity lays above the corresponding line of the decomposition. For our considerations, the native gate execution of the CCZ gate would be preferred for both hardware setups.

6.4. Shuttling

This section focuses on the comparison between the two mapping capabilities, namely long-range SWAP gates or qubit shuttling. Recently, Nottingham *et al* [32] have taken a first step toward qubit shuttling as a full replacement for SWAP gates. In the following, we want to study the possibilities of this approach in more detail. In particular, we want to study, how different hardware parameters like gate fidelity or coherence times affect the choice between the two alternatives. For this aim, we make multiple assumptions regarding the shuttling capability and analyze how the results compare to current gate-based shuttling approaches like SABRE [16].

First, we estimate the shuttling operations as a direct replacement for the computed SWAP gates and distinguish the two cases where shuttling can be executed simultaneously to regular quantum gates or has to be scheduled in a separate shuttling layer. As a second step, we examine the possible advantage of reconfiguring all qubits between different layers. Finally, we perform an error analysis on the shuttling velocity required for shuttling to be directly superior to a similar SWAP gate.

First, we make some assumptions about the shuttling error and the possibilities of parallelization for shuttling to simplify the analysis. For the gate-based shuttling approach, one has to consider errors stemming from imperfect CX gates as well as decoherence errors originating from qubit idling, as outlined in equation (16). In the case of perfect shuttling, only decoherence errors remain, as per equation (17). Our

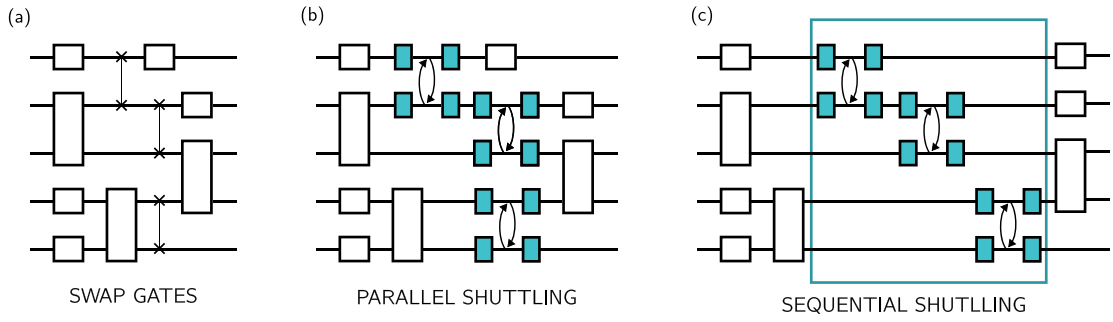

Figure 8. Shuttling Scenarios: (a) mapping output using a gate-based approach. (b) Parallel shuttling: the SWAP gates are replaced by shuttling operations in a one-to-one fashion. Each shuttling operation consists of initial trap-switch operations, the atom shuttling, and the final trap-switch operations. In this scenario, shuttling operations can be performed in parallel to each other and also to regular gate operations, requiring local qubit control at the hardware level. (c) Sequential shuttling: the same shuttling operations are performed but in a sequential order. Furthermore, regular gates and shuttling are executed in a non-parallel fashion, resulting in shuttling layers indicated by the cyan box.

interest here is in investigating a simplified model, wherein each SWAP operation between physical qubits Q_i and Q_j is replaced by a corresponding shuttling operation characterized by a duration approximated by:

$$t^{\mathrm{sh}}\left(\mathrm{SWAP}\left(Q_i, Q_j\right)\right) = 2\left(t_{\mathrm{trap}} + \frac{d\left(Q_i, Q_j\right)}{v_{\mathrm{s}}}\right). \tag{18}$$

In this equation, t_{trap} denotes the time required for transitioning between traps from SLM to AOD to initiate the shuttling and then back from AOD to SLM, d is the physical separation distance between the two qubits, and v_{s} stands for the maximal shuttling velocity. The factor of two takes into account the fact that the two shuttling operations required to swap the qubits have to be executed sequentially due to the non-cross shuttling constraint of equation (11). An illustration of this assumption is given in figure 12.

Considering the first hardware setup of Strontium atoms in table 1, we can estimate the shuttling time for a nearest neighbor SWAP operation as

$$t^{\mathrm{sh}} = 2\left(2 \cdot 20\,\mu\mathrm{s} + \frac{3\,\mu\mathrm{m}}{0.025\,\mu\mathrm{m}\,\mu\mathrm{s}^{-1}}\right) = 160\,\mu\mathrm{s}.$$

This is significantly faster than a corresponding gate-based SWAP consisting of three CX gates:

$$t_{\mathrm{SWAP}} \approx 3 \cdot (200\,\mu\mathrm{s} + 0.2\,\mu\mathrm{s}) \approx 600\,\mu\mathrm{s}.$$

As a result, for the Strontium setup, the shuttling SWAP operation is always preferable, as it is both faster and has a higher fidelity. Therefore, we will focus mainly on the second hardware setup based on Rubidium in the following.

6.4.1. Shuttling-based mapping

To draw a comparative analysis between the gate-based and shuttling-based mapping strategies, we intend to once again introduce variations in different hardware parameters favorable to one of the two cases. Specifically, we aim to delve into the influence of two parameters: the CX fidelity, denoted as $\mathcal{F}_{\mathrm{CX}}$, which substantially impacts the precision of gate-based shuttling, and, on the other side, the effective coherence time T_{eff}, corresponding to the decoherence errors occurring during shuttling. For the following evaluations, we always use a fixed circuit size of $n = 80$ qubits. This analysis maintains the assumption of equal values for r_{int} and r_{re}, fixed at d.

To perform our studies, we undertake circuit mapping using the SABRE compiler. Subsequently, following the considerations provided earlier, we engage in a post-processing step. This step encompasses the consolidation of consecutive SWAP gates, thereby generating the requisite shuttling operations, each with its corresponding shuttling distance. The duration for each shuttling operation is subsequently calculated following the formula delineated in equation (18).

Within the realm of scheduling, we consider two distinct scenarios, both depicted in figure 8 in comparison to a SWAP gate-based mapping.

In the first scenario, labeled as *parallel shuttling*, we presuppose the availability of a sufficiently high number of independent AODs. This premise allows for the concurrent execution of all shuttling operations

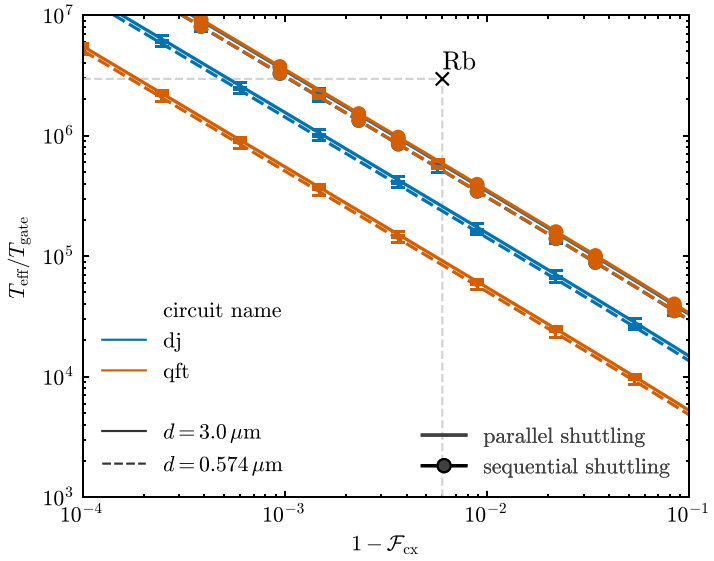

Figure 9. Error analysis for shuttling: a comparative evaluation of SWAP gate- and shuttling-based Mapping is presented. The coherence time at which the gate-based fidelity $\mathcal{F}_{\text{SWAP}}$ equates to the shuttling-based $\mathcal{F}_{\text{sh}}$ is showcased by aline, corresponding to two different circuits while varying the CX fidelities. The solid lines depict scenarios where shuttling and gate operations can be executed concurrently, while the line indicated with a bullet represents the case where shuttling and gate executions are performed alternately. The lines of sequential shuttling lay on top of each other, as due to the sequential execution of all shuttling operations the difference between circuit structures is reduced. Additionally, two different inter-qubit distances d are distinguished, indicated by solid and dashed lines. The error bars indicate the standard deviation averaged over 20 iterations.

that pertain to a non-overlapping set of qubits. Moreover, we assume that both gate and shuttling operations can be executed simultaneously, requiring local qubit control on the hardware level.

Conversely, in the second scenario, referred to as *sequential shuttling*, we operate under the constraint of possessing only a solitary AOD. Furthermore, the simultaneous execution of gate and shuttling operations is restricted, meaning shuttling and gate operations have to be executed in an alternating fashion, similar to the DPQA setting of figure 3. In this configuration, the circuit operation involves executing all gates feasible to the current qubit layout, succeeded by another layer encompassing the sequentially executed shuttling operations, resulting in *shuttling layers*, discussed in more detail in section 6.4.2.

It is important to note that the differentiation between these two approaches resides solely within the scheduling component, with the actual shuttling operations remaining unaltered.

Depicted in figure 9 is the boundary that delineates an equilibrium point wherein gate-based and shuttling-based methodologies yield precisely the same total fidelity. This assertion aligns with earlier discussions, signifying that data points situated above the depicted lines exhibit elevated coherence times and, therefore, favor a shuttling-based approach. Notably, higher CX fidelities necessitate elevated values of T_{eff} to render shuttling-based mapping competitive.

Within the context of parallel shuttling, as denoted by the continuous lines, a difference between different circuits is visible. Circuits inherently characterized by a sequential architecture, exemplified by cases like the Deutsch Jozsa algorithm (blue), diverge from circuits with more parallel structures, such as the Quantum Fourier Transform(red), which benefits from the concurrent execution of shuttling operations. Conversely, when confined to a sole AOD, as indicated by the circled lines, disparities between these categories tend to attenuate, as in this case, all operations are executed sequentially. In addition, we varied the inter-qubit distance d between the value $d = 3\,\mu$m as given in table 1 and a prospective value of $d = 0.574\,\mu$m. For smaller d (dashed line) lower T_{eff} are required, as the shuttling takes less time. Nevertheless, the difference is small, as SABRE accounts only for nearest neighbor SWAPS and in this case, the shuttling duration is almost neglectable compared to the trap switching time of $t_{\text{trap}} = 40\,\mu$m.
Regarding the Rubidium hardware configuration, the large T_{eff} favors shuttling-based swapping in all cases compared to the error-prone gate-based swapping. CX fidelities of $\mathcal{F}_{\text{cx}} \approx 0.999$ would be necessary to make gate-based swapping a comparable alternative again.

Nonetheless, as this approach uses the SC-specific SABRE algorithm to find necessary SWAP gates, it does not take full advantage of the shuttling capability, which can also perform non-local and non-trivial SWAP operations. We will take account of this shortcoming in the following section.

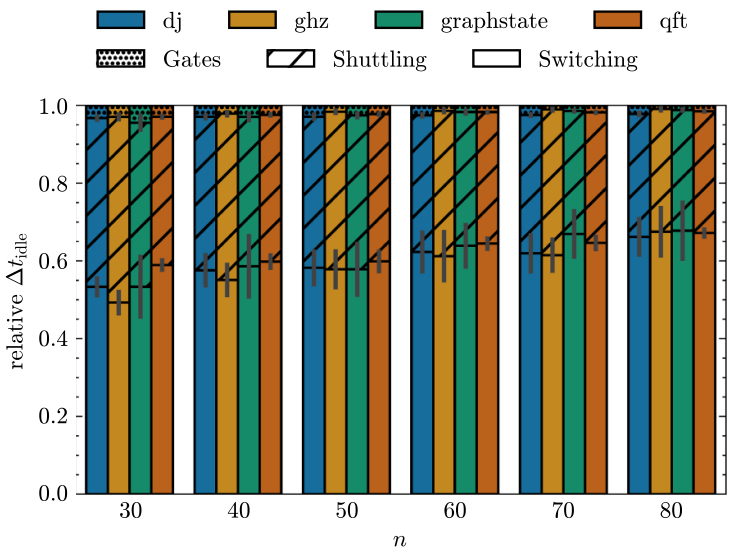

Figure 10. Comparative analysis of idle times: the correlation between shuttling and gate execution times in layer-based shuttling is depicted as a ratio. This ratio is influenced by distinct circuit types and increasing circuit sizes (n). The analysis takes into account the Rubidium hardware configuration for both shuttling and gate timings. Remarkably, over 90% of the overall circuit execution time is attributed to shuttling and trap-switching operations, with this proportion rising as the number of qubits increases. The large contribution of gate-switch is caused by SABRE preferring multiple short-range SWAP gates, each requiring the corresponding trap switch duration t_{trap}. The error bars indicate the standard deviation over 50 iterations.

6.4.2. Shuttling layers

Now, we want to consider the second shuttling scenario of sequential, layer-based shuttling. This separation between gate execution and shuttling facilitates the examination of their respective impacts on the overall idle time, denoted as t_{idle}, and thereby, the resultant decoherence error.

As depicted in figure 10, the graph portrays the proportion of idling attributed to gate-related factors versus idling linked to shuttling operations across different circuits and varying numbers of qubits. For all instances considered, the contribution from shuttling operations accounts for more than 90% of the cumulative idle time, in particular for larger qubit numbers. Furthermore, it shows that for our assumptions more than half of the total idle time is due to trap switching. Here, it should be noted that the long trap switching times are due to SABRE, with its SC background, preferring multiple short-distance SWAPs compared to a single long-range operation. So the results can be expected to be improved by employing a shuttling-specific compiler.

This method, involving the utilization of SABRE followed by the subsequent substitution of resultant SWAPs with shuttling operations, does, therefore, not use the full shuttling potential. SABRE consistently opts for the shortest feasible swapping path to minimize the count of SWAP gates. Conversely, for shuttling, it may be strategically advantageous to transport a qubit across a greater distance, potentially improving the future requirement for swapping. To address this prospect, we modify the mapping process, whereby SABRE is initially employed to ascertain a mapping configuration, enabling the execution of as many gates as feasible. Subsequently, a wholly fresh initial mapping is determined for the remaining circuit. Then again, all feasible gates are executed, and the process of finding a new mapping for the next layer is iterated until the complete circuit is processed.

As depicted in figure 11, the illustration showcases the proportion by which the cumulative count of layers can be diminished by employing this comprehensive qubit reconfiguration for each layer. Depending on the specific circuit and the count of qubits, this approach can yield reductions of up to 50% in the number of shuttling layers. While it is generally expected that the required inter-layer shuttling becomes more complicated with this technique, it still underscores a promising benefit of utilizing the full reconfiguration capabilities of NAQC platforms.

Until now, we assumed fixed shuttling hardware parameters and only modified T_{eff}. But also other parameters such as the trap switching time or the actual shuttling velocity v_{sh} affect the shuttling time of equation (18) and therefore the shuttling error. In the following, we want to study these parameters by the question, how fast must the shuttling be to be favorable compared to a corresponding gate-based SWAP operation?

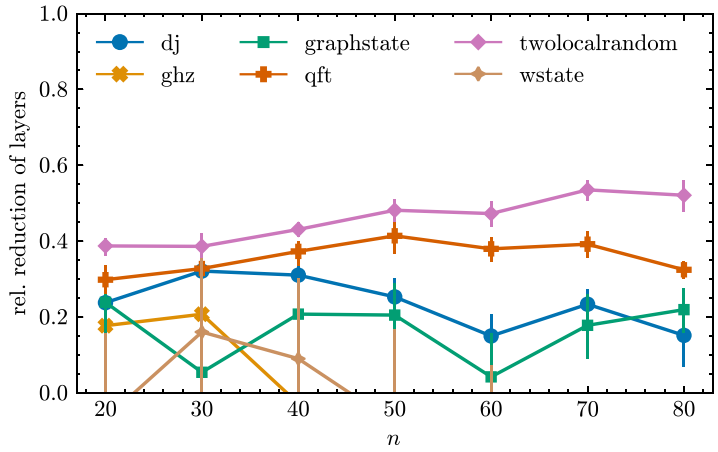

Figure 11. Layer reduction via full reconfiguration: the lines present the ratio by which the number of layers is reduced when the mapping algorithm is granted complete reconfigurability for each layer. This demonstrates the potential improvement by utilizing long-range shuttling compared to nearest neighbor SWAPs. The error bars indicate the standard deviation over 50 iterations.

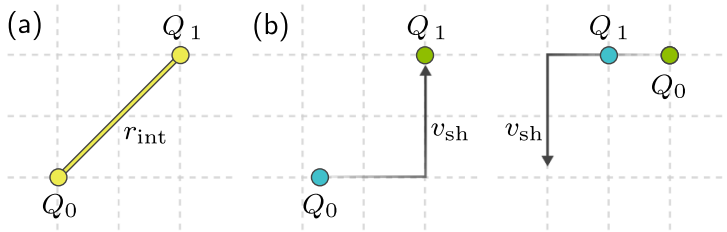

Figure 12. Shuttling as direct SWAP substitution: illustration of the estimations taken in figure 13. (a) SWAP(Q_0, Q_1) executed as a long-range interaction over distance r_{int}. (b) Illustration of a two-step shuttling process for the same SWAP operation, using shuttling. Due to equation (11) these operations can not be executed exactly as shown, but are used as a first approximation of the actually required operations.

6.4.3. Shuttling velocity

The shuttling duration, as expressed in equation (18), is determined by two components: the time t_{trap} required to transition between trap types, i.e. SLM to AOD and vice versa, and the shuttling velocity v_{sh}. When considering a specific CX fidelity $\mathcal{F}_{\mathrm{CX}}$, the duration of shuttling must be less than a certain critical value for shuttling-based SWAP to exhibit improved error performance compared to the equivalent SWAP gate.

In the context of the Rubidium hardware, even though the time taken for shuttling-based SWAP surpasses that of three consecutive CX gates, the large T_{eff} mitigates decoherence errors, particularly when only two qubits are involved. However, with larger values of n, which also implies more qubits idling during the shuttling SWAP, a crossover exists where less accurate yet faster CX gates become the preferred choice. This is of particular interest for SWAP operations that can not be parallelized with other gate operations, occurring often, particularly for sequentially structured algorithms.

An illustration of this direct substitution of gate-based SWAP operations by the corresponding shuttling operations is shown in figure 12, where the SWAP(Q_0, Q_1) gate over a distance r_{int} is substituted by two shuttling operations.

Figure 13 showcases the required shuttling velocity (v_{sh}) to achieve parity of $\mathcal{F}_{\mathrm{SWAP}}$ and $\mathcal{F}_{\mathrm{sh}}$ for a single SWAP operation, depending upon the count of idle qubits. Points situated above the lines denote scenarios where the implementation of shuttling for SWAP gates results in improved shuttling fidelity compared to the gate-based alternative. Within the context of the Rubidium scenario ($\mathcal{F}_{\mathrm{CX}} = 0.995$ and $r_{\mathrm{int}} = 2d$, shown in blue), the critical number of idle qubits is determined by the intersection with the maximal shuttling velocity ($v_{\mathrm{sh}} = 0.55\,\mu\mathrm{m}\,\mu\mathrm{s}^{-1}$), occurring at approximately 550 qubits. As examples, higher CX fidelities and interaction radii are depicted, both reducing further the maximal number of allowed idle qubits. While this consideration does not take into account parallel shuttling, it still shows the possibility of preferring SWAP gates compared to shuttling for near-term hardware with qubit numbers in the range of multiple hundreds.

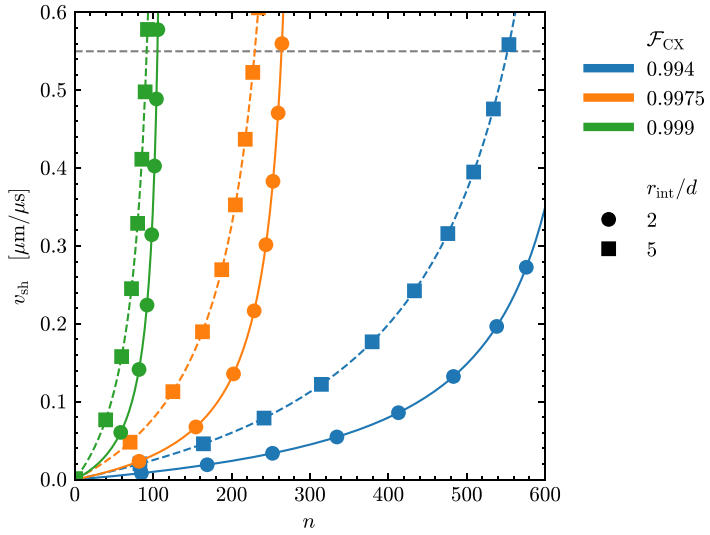

Figure 13. Shuttling velocities: depicted here is the minimum necessary shuttling velocity to attain an error equivalent to that induced by gate-based swapping for a single swap pair for the Rubidium hardware setting. The x-axis indicates the number of qubits n which must idle during the shuttling operation. For more idle qubits, and therefore increased idle errors, the shuttling speed must also increase to keep the overall shuttling error the same. The steep increase of v_{sh} is due to the fact that the shuttling time has a lower limit by the trap switching time t_{trap}. For an increased interaction radius r_{int}, the shuttling has to be performed faster due to the longer shuttling distance. While shuttling is a suitable substitute for SWAP gates with a current fidelity of 0.995, even for high idle qubit numbers of 500, an increase in $\mathcal{F}_{\mathrm{CX}}$ favors the gate-based swapping again.

The advantages of a shuttling-based approach for shuttling over longer distances and in parallel will be subject to future work, requiring a fully working shuttling compiler.

6.4.4. Discussion—shuttling

The approximations conducted in this section underscore the substantial promise inherent to atom shuttling as a prospective substitute for gate-based mapping. This particularly holds true for scenarios characterized by large T_{eff} and lower $\mathcal{F}_{\mathrm{CX}}$. These insights can be beneficial for hardware experts striving to enhance available hardware capabilities. Considerations like the layer-based complete reconfiguration, as demonstrated in figure 11, underscore the latent advantages of employing mapping strategies specifically tailored to shuttling.

For compiler developers, this introduces a range of implications. Initially, compilation strategies must ascertain the availability of shuttling operations, and subsequently devise mapping strategies that effectively leverage the capacity to perform qubit swaps over significant distances in parallel. Recently, pioneering work in this direction has been made [32, 116]. Our calculations indicate, furthermore, that for the Rubidium hardware setting with a single AOD, the shuttling and trap switch process constitutes the central part of qubit idling. Additionally, we show that using the capability of NAs to completely rearrange all qubits after each gate layer can reduce the total number of shuttling layers by up to 50%. In this way, we delineate the prospective advantage of adopting a fully reconfigurable mapping strategy instead of the conventional nearest-neighbor methodology.

Furthermore, the evaluations presented in figure 13 offer indications that hardware parameters such as T_{eff}, $\mathcal{F}_{\mathrm{CX}}$, r_{int}, and the number of idling qubits n impact the decision of whether a SWAP operation should be executed using the slower yet accurate shuttling method or faster gate operations, although error-prone. This accentuates the need for a hybrid compiler strategy capable of making informed choices based on the underlying hardware parameters, guiding the execution of remapping through either of the two available capabilities. For decision-making, a similar approach as proposed in this work can be employed to find the optimal compilation process, given the available hardware.

6.5. Discussion

In this section, we performed multiple selected case studies using different hardware parameters to study the compilation output error for the available NA capabilities.

For the possibility of long-range SWAP gates, we focused on comparing fidelity reduction due to SWAP gates and decoherence on the other hand. Our considerations indicate that for the Rubidium setup, the SWAP gate errors dominate for all considered circuits. In this scenario, high connectivity with $r_{\mathrm{int}} \geqslant 5d$

would be necessary to get comparable error contributions from decoherence errors. This also implies that SWAP gate minimization can be considered a suitable figure of merit, making common SC compilers such as SABRE and their techniques interesting candidates. For the first hardware setup, based on Strontium atoms, on the other hand, both errors contribute equally, requiring simultaneous optimization of the SWAP gate and idle-time minimization. This implies the need for hardware adaptive compilers that can optimize different figures of merit depending on given hardware parameters and, therefore, the primary source of errors.

For multi-qubit gates, the task of synthesis remains an open question. In contrast, a simple error estimation can be performed for circuits already containing multi-qubit gates to decide if the gate should be executed natively or decomposed.

For the shuttling capability, we make multiple simplifying assumptions to illustrate the promising potential as an alternative to regular gate-based swapping. We compare the two swapping techniques concerning different hardware parameters such as coherence times, CX gate fidelity, shuttling velocity, and qubit number. As a result, shuttling outperforms gate-based swapping for both considered hardware setups. Nevertheless, we also indicate possible situations regarding high CX fidelity or a large number of idle qubits, where the error-prone but faster SWAP gates become interesting again. This would imply the need for a hybrid compilation process, where the mapping pass decides dynamically if gate-based swapping or shuttling-based qubit rearrangement is favorable.

In summary, the evaluations performed in this section illustrate the use of the capabilities discussed previously and give multiple insights on how different hardware parameters affect the compilation output. This is helpful for tool developers to build hardware-aware compilation software based on optimization techniques based on valuable figures of merit and, this way, facilitates the development of NA-specific compilers. At the same time, the results can also give hardware experts insight into devising future hardware, prioritizing hardware attributes that yield the most likely output improvement. As an advantage, this allows for the effective co-design of hardware setups and compilation software, necessary to explore the full capabilities of the NAQC platform.

7. Summary and outlook

In this work, we studied the overall compiler development for the Neutral Atom Quantum Computing platform and provided important groundwork to promote further collaboration between hardware experts and computer scientists.

Initially, we expounded upon the foundational physical aspects integral to realizing quantum processors utilizing neutral atoms, explicitly emphasizing the distinct computational capabilities intrinsic to the platform. We provided abstraction layers that cater to the design automation community and software tool developers, facilitating their comprehension and abstraction of the physical processes. Subsequently, we delved into the structural organization of this spectrum of compilation strategies and explored figures of merit for assessing the quality of the resultant compilation outcomes. Furthermore, we furnished an overarching view of the currently available software tools and compilers, contextualizing their roles within the previously discussed compilation overview. Lastly, we performed multiple selective case studies and fidelity analyses to investigate the implications of different hardware parameters on the final compilation outcome. In particular, we evaluate and compare the different capabilities and match the results to two possible hardware setups, giving insights to both hardware experts and tool developers. The results underscore the imperative to develop hybrid and hardware-aware compilation software capable of effectively addressing the broad spectrum of capabilities offered by the neutral atom platform.

We posit that this comprehensive overview can effectively contribute to the development of top-tier compilers and design automation tools, enabling the neutral atom platform to catch up with comparable solutions available for other hardware platforms. In particular, the next steps entail the development of new compilation software, based on the information gained within this work regarding possible hardware capabilities and useful figures of merit.

Furthermore, neutral atoms have been shown to be promising candidates for fault-tolerant quantum computing [8, 100] due to their extended range of capabilities. This includes high gate fidelities, combined with e.g. the ability to perform native multi-qubit gates, mid-circuit measurements, and qubit shuttling. In particular, the qubit shuttling allows beyond nearest-neighbor connectivity without breaking fault tolerance due to error propagation such as SWAP gates, and the simultaneous control of large qubit patches using parallel AOD movements. The fundamental basics discussed in this work also pave the way toward fault-tolerant compilation, which will add another layer of complexity to the compilation chain, and finding suitable automation techniques is still an open question.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://doi.org/10.5281/zenodo.8347713.

Acknowledgments

All authors acknowledge funding from the Munich Quantum Valley initiative (K3, K5, K8), which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus. L S and R W acknowledge funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant Agreement No. 101001318). D F L, and M M acknowledge support by the BMBF project IQuAn. M R and M M also acknowledge funding by the Deutsche Forschungsgemeinschaft under Germany's Excellence Strategy 'Cluster of Excellence Matter and Light for Quantum Computing (ML4Q)' EXC-2004/1-390534769. J Z and S B acknowledge funding by the Max Planck Society (MPG), the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy EXC-2111-39081486. J Z acknowledges support from the BMBF through the program 'Quantum technologies—from basic research to market' (SNAQC, Grant No. 13N16265). D F L, M M, J Z and S B additionally acknowledge support by the BMBF Project MUNIQC-ATOMS (Grant No. 13N16070). S B and J Z are co-founders and shareholders of planqc.

ORCID iDs

Ludwig Schmid https://orcid.org/0000-0002-4246-8125
David F Locher https://orcid.org/0009-0001-0180-4225
Sebastian Blatt https://orcid.org/0000-0003-2466-9967
Johannes Zeiher https://orcid.org/0000-0002-8466-1863
Markus Müller https://orcid.org/0000-0002-2813-3097
Robert Wille https://orcid.org/0000-0002-4993-7860

References

[1] Saffman M 2016 Quantum computing with atomic qubits and Rydberg interactions: progress and challenges *J. Phys. B: At. Mol. Opt. Phys.* **49** 202001
[2] Morgado M and Whitlock S 2021 Quantum simulation and computing with Rydberg-interacting qubits *AVS Quantum Sci.* **3** 023501
[3] Graham T M *et al* 2022 Multi-qubit entanglement and algorithms on a neutral-atom quantum computer *Nature* **604** 457
[4] Levine H *et al* 2019 Parallel implementation of high-fidelity multiqubit gates with neutral atoms *Phys. Rev. Lett.* **123** 170503
[5] Evered S J *et al* 2023 High-fidelity parallel entangling gates on a neutral-atom quantum computer *Nature* **622** 268
[6] Müller M, Lesanovsky I, Weimer H, Büchler H P and Zoller P 2009 Mesoscopic Rydberg gate based on electromagnetically induced transparency *Phys. Rev. Lett.* **102** 170502
[7] Isenhower L, Saffman M and Mølmer K 2011 Multibit CkNOT quantum gates via Rydberg blockade *Quantum Inf. Process.* **10** 755
[8] Bluvstein D *et al* 2023 Logical quantum processor based on reconfigurable atom arrays *Nature* **626** 58–65
[9] Bluvstein D *et al* 2022 A quantum processor based on coherent transport of entangled atom arrays *Nature* **604** 451
[10] Ebadi S *et al* 2022 Quantum optimization of maximum independent set using Rydberg atom arrays *Science* **376** 1209
[11] Barredo D, de Léséleuc S, Lienhard V, Lahaye T and Browaeys A 2016 An atom-by-atom assembler of defect-free arbitrary two-dimensional atomic arrays *Science* **354** 1021
[12] Endres M, Bernien H, Keesling A, Levine H, Anschuetz E R, Krajenbrink A, Senko C, Vuletic V, Greiner M and Lukin M D 2016 Atom-by-atom assembly of defect-free one-dimensional cold atom arrays *Science* **354** 1024
[13] Qiskit Contributors 2023 Qiskit: an open-source framework for quantum computing *Zenodo* https://doi.org/10.5281/zenodo.2573505
[14] Wille R and Burgholzer L 2023 MQT QMAP: efficient quantum circuit mapping *Proc. 2023 Int. Symp. on Physical Design (ISPD '23)* (Association for Computing Machinery) pp 198–204
[15] Cowtan A, Dilkes S, Duncan R, Krajenbrink A, Simmons W and Sivarajah S 2019 On the qubit routing problem *14th Conf. on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)* (*Leibniz Int. Proc. in Informatics (LIPIcs)* vol 135) ed W van Dam and L Mancinska (Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik) pp 5:1–5:32
[16] Li G, Ding Y and Xie Y 2019 Tackling the qubit mapping problem for NISQ-era quantum devices *Proc. 24th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)* (Association for Computing Machinery) pp 1001–14
[17] Zulehner A, Paler A and Wille R 2019 An efficient methodology for mapping quantum circuits to the IBM QX architectures *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **38** 1226
[18] Bergholm V *et al* 2022 PennyLane: automatic differentiation of hybrid quantum-classical computations (arXiv:1811.04968)
[19] Cirq Developers 2023 Cirq *Zenodo* https://doi.org/10.5281/zenodo.8161252
[20] Steiger D S, Häner T and Troyer M 2018 ProjectQ: an open source software framework for quantum computing *Quantum* **2** 49
[21] Tan B and Cong J 2020 Optimal layout synthesis for quantum computing *Proc. 39th Int. Conf. on Computer-Aided Design (ICCAD '20)* (Association for Computing Machinery) pp 1–9

[22] Sivarajah S, Dilkes S, Cowtan A, Simmons W, Edgington A and Duncan R 2020 t|ket⟩: a retargetable compiler for NISQ devices *Quantum Sci. Technol.* **6** 014003
[23] Saki A A, Topaloglu R O and Ghosh S 2022 Muzzle the shuttle: efficient compilation for multi-trap trapped-ion quantum computers *2022 Design, Automation & Test in Europe Conf. & Exhibition (DATE)* pp 322–7
[24] Kreppel F, Melzer C, Millán D O, Wagner J, Hilder J, Poschinger U, Schmidt-Kaler F and Brinkmann A 2023 Quantum circuit compiler for a shuttling-based trapped-ion quantum computer *Quantum* **7** 1176
[25] Schmale T, Temesi B, Baishya A, Pulido-Mateo N, Krinner L, Dubielzig T, Ospelkaus C, Weimer H and Borcherding D 2022 Backend compiler phases for trapped-ion quantum computers *2022 IEEE Int. Conf. on Quantum Software (QSW)* pp 32–37
[26] Maslov D 2017 Basic circuit compilation techniques for an ion-trap quantum machine *New J. Phys.* **19** 023035
[27] Schoenberger D, Hillmich S, Brandl M and Wille R 2023 Using Boolean satisfiability for exact shuttling in trapped-ion quantum computers *29th Asia and South Pacific Design Automation Conference* (*Incheon Songdo Convensia, South Korea, 22–25 January 2024)* (arXiv:2311.03454 [quant-ph])
[28] Li Y, Zhang Y, Chen M, Li X and Xu P 2023 Timing-aware qubit mapping and gate scheduling adapted to neutral atom quantum computing *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **42** 3768–80
[29] Patel T, Silver D and Tiwari D 2022 Geyser: a compilation framework for quantum computing with neutral atoms *Proc. 49th Annual Int. Symp. on Computer Architecture (ISCA '22)* (Association for Computing Machinery) pp 383–95
[30] Tan B, Bluvstein D, Lukin M D and Cong J 2022 Qubit mapping for reconfigurable atom arrays *Proc. 41st IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD '22)* (Association for Computing Machinery) pp 1–9
[31] Brandhofer S, Polian I and Büchler H P 2021 Optimal mapping for near-term quantum architectures based on Rydberg atoms *2021 IEEE/ACM Int. Conf. On Computer Aided Design (ICCAD)* pp 1–7
[32] Nottingham N, Perlin M A, White R, Bernien H, Chong F T and Baker J M 2023 Decomposing and routing quantum circuits under constraints for neutral atom architectures (arXiv:2307.14996)
[33] Baker J M, Litteken A, Duckering C, Hoffmann H, Bernien H and Chong F T 2021 Exploiting long-distance interactions and tolerating atom loss in neutral atom quantum architectures *2021 ACM/IEEE 48th Annual Int. Symp. on Computer Architecture (ISCA)* pp 818–31
[34] Tan D B, Ping S and Cong J 2024 Depth-optimal addressing of 2D qubit array with 1D controls based on exact binary matrix factorization (arXiv:2401.13807)
[35] Wang H, Liu P, Tan B, Liu Y, Gu J, Pan D Z, Cong J, Acar U and Han S 2023 FPQA-C: a compilation framework for field programmable qubit array (arXiv:2311.15123)
[36] Schmid L, Park S, Kang S and Wille R 2023 Hybrid circuit mapping: leveraging the full spectrum of computational capabilities of neutral atom quantum computers (arXiv:2311.14164)
[37] Chong F T, Franklin D and Martonosi M 2017 Programming languages and compiler design for realistic quantum hardware *Nature* **549** 180
[38] JavadiAbhari A, Patil S, Kudrow D, Heckey J, Lvov A, Chong F T and Martonosi M 2014 ScaffCC: a framework for compilation and analysis of quantum computing programs *Proc. 11th ACM Conf. on Computing Frontiers (CF '14)* (Association for Computing Machinery) pp 1–10
[39] Kitaev A Y 1997 Quantum computations: algorithms and error correction *Russ. Math. Surv.* **52** 1191
[40] Saffman M, Walker T G and Mølmer K 2010 Quantum information with Rydberg atoms *Rev. Mod. Phys.* **82** 2313
[41] Adams C S, Pritchard J D and Shaffer J P 2019 Rydberg atom quantum technologies *J. Phys. B: At. Mol. Opt. Phys.* **53** 012002
[42] Henriet L, Beguin L, Signoles A, Lahaye T, Browaeys A, Reymond G-O and Jurczak C 2020 Quantum computing with neutral atoms *Quantum* **4** 327
[43] Wu X, Liang X, Tian Y, Yang F, Chen C, Liu Y-C, Tey M K and You L 2021 A concise review of Rydberg atom based quantum computation and quantum simulation* *Chin. Phys.* B **30** 020305
[44] Levine H, Keesling A, Omran A, Bernien H, Schwartz S, Zibrov A S, Endres M, Greiner M, Vuletićć V and Lukin M D 2018 High-fidelity control and entanglement of Rydberg-atom qubits *Phys. Rev. Lett.* **121** 123603
[45] DiVincenzo D P 2000 The physical implementation of quantum computation *Fortschr. Phys.* **48** 771
[46] Grimm R, Weidemüller M and Ovchinnikov Y B 2000 *Optical Dipole Traps for Neutral Atoms* (Academic) pp 95–170
[47] Jessen P and Deutsch I 1996 *Optical Lattices* (Academic) pp 95–138
[48] Kaufman A M and Ni K-K 2021 Quantum science with optical tweezer arrays of ultracold atoms and molecules *Nat. Phys.* **17** 1324
[49] Gyger F, Ammenwerth M, Tao R, Timme H, Snigirev S, Bloch I and Zeiher J 2024 Continuous operation of large-scale atom arrays in optical lattices (arXiv:2402.04994)
[50] Norcia M A *et al* 2024 Iterative assembly of ^{171}Yb atom arrays in cavity-enhanced optical lattices (arXiv:2401.16177)
[51] Kaufman A M, Lester B J and Regal C A 2012 Cooling a single atom in an optical tweezer to its quantum ground state *Phys. Rev.* X **2** 041014
[52] Graham T M, Kwon M, Grinkemeyer B, Marra Z, Jiang X, Lichtman M T, Sun Y, Ebert M and Saffman M 2019 Rydberg-mediated entanglement in a two-dimensional neutral atom qubit array *Phys. Rev. Lett.* **123** 230501
[53] Barnes K *et al* 2022 Assembly and coherent control of a register of nuclear spin qubits *Nat. Commun.* **13** 2779
[54] Ma S, Burgers A P, Liu G, Wilson J, Zhang B and Thompson J D 2022 universal gate operations on nuclear spin qubits in an optical tweezer array of ^{171}Yb atoms *Phys. Rev.* X **12** 021028
[55] Jenkins A, Lis J W, Senoo A, McGrew W F and Kaufman A M 2022 Ytterbium nuclear-spin qubits in an optical tweezer array *Phys. Rev.* X **12** 021027
[56] Young A W, Eckner W J, Milner W R, Kedar D, Norcia M A, Oelker E, Schine N, Ye J and Kaufman A M 2020 Half-minute-scale atomic coherence and high relative stability in a tweezer clock *Nature* **588** 408
[57] Schine N, Young A W, Eckner W J, Martin M J and Kaufman A M 2022 Long-lived Bell states in an array of optical clock qubits *Nat. Phys.* **18** 1067
[58] Pucher S, Klüsener V, Spriestersbach F, Geiger J, Schindewolf A, Bloch I and Blatt S 2024 Fine-structure qubit encoded in metastable strontium trapped in an optical lattice (arXiv:2401.11054)
[59] Unnikrishnan G *et al* 2024 Coherent control of the fine-structure qubit in a single alkaline-earth atom (arXiv:2401.10679)
[60] Hölzl C, Götzelmann A, Pultinevicius E, Wirth M and Meinert F 2024 Long-lived circular Rydberg qubits of alkaline-earth atoms in optical tweezers (arXiv:2401.10625)
[61] Anand S, Bradley C E, White R, Ramesh V, Singh K and Bernien H 2024 A dual-species Rydberg array (arXiv:2401.10325)
[62] Gallagher T F 1994 *Rydberg Atoms* (*Cambridge Monographs on Atomic, Molecular and Chemical Physics*) (Cambridge University Press)

[63] Jaksch D, Cirac J I, Zoller P, Rolston S L, Côté R and Lukin M D 2000 Fast quantum gates for neutral atoms *Phys. Rev. Lett.* **85** 2208
[64] Madjarov I S, Covey J P, Shaw A L, Choi J, Kale A, Cooper A, Pichler H, Schkolnik V, Williams J R and Endres M 2020 High-fidelity entanglement and detection of alkaline-earth Rydberg atoms *Nat. Phys.* **16** 857
[65] Wang G, Xu W, Li C, Vuletić V and Cappellaro P 2023 Individual-atom control in array through phase modulation (arXiv:2310.19741)
[66] Isenhower L, Urban E, Zhang X L, Gill A T, Henage T, Johnson T A, Walker T G and Saffman M 2010 Demonstration of a neutral atom controlled-NOT quantum gate *Phys. Rev. Lett.* **104** 010503
[67] Wilk T, Gaëtan A, Evellin C, Wolters J, Miroshnychenko Y, Grangier P and Browaeys A 2010 Entanglement of two individual neutral atoms using Rydberg blockade *Phys. Rev. Lett.* **104** 010502
[68] Dlaska C, Ender K, Mbeng G B, Kruckenhauser A, Lechner W and van Bijnen R 2022 Quantum optimization via four-body Rydberg gates *Phys. Rev. Lett.* **128** 120503
[69] Shende V V, Prasad A K, Markov I L and Hayes J P 2002 Reversible logic circuit synthesis *Proc. 2002 IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD '02)* (Association for Computing Machinery) pp 353–60
[70] Adarsh S, Burgholzer L, Manjunath T and Wille R 2022 SyReC synthesizer: an MQT tool for synthesis of reversible circuits *Softw. Impacts* **14** 100451
[71] Wille R, Große D, Teuber L, Dueck G W and Drechsler R 2008 RevLib: an online resource for reversible functions and reversible circuits *38th Int. Symp. on Multiple Valued Logic (ismvl 2008)* pp 220–5
[72] Wille R, Offermann S and Drechsler R 2010 SyReC: a programming language for synthesis of reversible circuits *2010 Forum on Specification & Design Languages (FDL 2010)* pp 1–6
[73] Amy M, Glaudell A N, Li S M and Ross N J 2023 Improved Synthesis of Toffoli-Hadamard Circuits *Reversible Computation* (*Lecture Notes in Computer Science*) ed M Kutrib and U Meyer (Springer Nature Switzerland) pp 169–209
[74] Aharonov D 2003 A simple proof that Toffoli and Hadamard are quantum universal (arXiv:0301040)
[75] Beugnon J *et al* 2007 Two-dimensional transport and transfer of a single atomic qubit in optical tweezers *Nat. Phys.* **3** 696
[76] Kwon M, Ebert M F, Walker T G and Saffman M 2017 Parallel low-loss measurement of multiple atomic qubits *Phys. Rev. Lett.* **119** 180504
[77] Covey J P, Madjarov I S, Cooper A and Endres M 2019 2000-times repeated imaging of strontium atoms in clock-magic tweezer arrays *Phys. Rev. Lett.* **122** 173201
[78] Bochmann J, Mücke M, Guhl C, Ritter S, Rempe G and Moehring D L 2010 Lossless state detection of single neutral atoms *Phys. Rev. Lett.* **104** 203601
[79] Deist E, Lu Y-H, Ho J, Pasha M K, Zeiher J, Yan Z and Stamper-Kurn D M 2022 Mid-circuit cavity measurement in a neutral atom array *Phys. Rev. Lett.* **129** 203602
[80] Graham T M, Phuttitarn L, Chinnarasu R, Song Y, Poole C, Jooya K, Scott J, Scott A, Eichler P and Saffman M 2023 Midcircuit measurements on a single-species neutral alkali atom quantum processor *Phys. Rev.* X **13** 041051
[81] Norcia M A *et al* 2023 Midcircuit qubit measurement and rearrangement in a ^{171}Yb atomic array *Phys. Rev.* X **13** 041034
[82] Lis J W, Senoo A, McGrew W F, Rönchen F, Jenkins A and Kaufman A M 2023 Midcircuit operations using the *omg* architecture in neutral atom arrays *Phys. Rev.* X **13** 041035
[83] Huie W, Li L, Chen N, Hu X, Jia Z, Sun W K C and Covey J P 2023 Repetitive readout and real-time control of nuclear spin qubits in ^{171}Yb atoms *PRX Quantum* **4** 030337
[84] Singh K, Bradley C E, Anand S, Ramesh V, White R and Bernien H 2023 Mid-circuit correction of correlated phase errors using an array of spectator qubits *Science* **380** 1265
[85] Pagano A, Weber S, Jaschke D, Pfau T, Meinert F, Montangero S and Büchler H P 2022 Error budgeting for a controlled-phase gate with strontium-88 Rydberg atoms *Phys. Rev. Res.* **4** 033019
[86] Jandura S and Pupillo G 2022 Time-optimal two- and three-qubit gates for Rydberg atoms *Quantum* **6** 712
[87] Campbell E T, Terhal B M and Vuillot C 2017 Roads towards fault-tolerant universal quantum computation *Nature* **549** 172
[88] Cong I, Levine H, Keesling A, Bluvstein D, Wang S-T and Lukin M D 2022 Hardware-efficient, fault-tolerant quantum computation with Rydberg atoms *Phys. Rev.* X **12** 021049
[89] Wu Y, Kolkowitz S, Puri S and Thompson J D 2022 Erasure conversion for fault-tolerant quantum computing in alkaline earth Rydberg atom arrays *Nat. Commun.* **13** 4657
[90] Sahay K, Jin J, Claes J, Thompson J D and Puri S 2023 High threshold codes for neutral atom qubits with biased erasure errors *Phys. Rev.* X **13** 041013
[91] Scholl P, Shaw A L, Tsai R B-S, Finkelstein R, Choi J and Endres M 2023 Erasure conversion in a high-fidelity Rydberg quantum simulator *Nature* **622** 273
[92] Ma S, Liu G, Peng P, Zhang B, Jandura S, Claes J, Burgers A P, Pupillo G, Puri S and Thompson J D 2023 High-fidelity gates and mid-circuit erasure conversion in an atomic qubit *Nature* **622** 279
[93] Jandura S, Thompson J D and Pupillo G 2023 Optimizing Rydberg gates for logical-qubit performance *PRX Quantum* **4** 020336
[94] Fromonteil C, Bluvstein D and Pichler H 2023 Protocols for Rydberg entangling gates featuring robustness against quasistatic errors *PRX Quantum* **4** 020335
[95] Heußen S, Locher D F and Müller M 2024 Measurement-free fault-tolerant quantum error correction in near-term devices *PRX Quantum* **5** 010333
[96] Crow D, Joynt R and Saffman M 2016 Improved error thresholds for measurement-free error correction *Phys. Rev. Lett.* **117** 130503
[97] Perlin M A, Premakumar V N, Wang J, Saffman M and Joynt R 2023 Fault-tolerant measurement-free quantum error correction with multiqubit gates *Phys. Rev.* A **108** 062426
[98] Nagib O, Huft P, Safari A and Saffman M 2023 Robust atom-photon gate for quantum information processing (arXiv:2312.13221)
[99] Li Y and Thompson J 2024 High-rate and high-fidelity modular interconnects between neutral atom quantum processors (arXiv:2401.04075)
[100] Xu Q, Ataides J P B, Pattison C A, Raveendran N, Bluvstein D, Wurtz J, Vasic B, Lukin M D, Jiang L and Zhou H 2023 Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays (arXiv:2308.08648)
[101] Viszlai J, Lin S F, Dangwal S, Baker J M and Chong F T 2023 An architecture for improved surface code connectivity in neutral atoms (arXiv:2309.13507)
[102] Viszlai J, Yang W, Lin S F, Liu J, Nottingham N, Baker J M and Chong F T 2023 Matching generalized-bicycle codes to neutral atoms for low-overhead fault-tolerance (arXiv:2311.16980)

[103] Wang Y-F, Wang Y, Chen Y-A, Zhang W, Zhang T, Hu J, Chen W, Gu Y and Liu Z-W 2023 Efficient fault-tolerant implementations of non-Clifford gates with reconfigurable atom arrays (arXiv:2312.09111)
[104] Delfosse N, Beverland M E and Tremblay M A 2021 Bounds on stabilizer measurement circuits and obstructions to local implementations of quantum LDPC codes (arXiv:2109.14599)
[105] Tremblay M A, Delfosse N and Beverland M E 2022 Constant-overhead quantum error correction with thin planar connectivity *Phys. Rev. Lett.* **129** 050504
[106] Strikis A and Berent L 2023 Quantum low-density parity-check codes for modular architectures *PRX Quantum* **4** 020321
[107] Cross Andrew*et al* 2022 OpenQASM 3: a broader and deeper quantum assembly language—ACM transactions on quantum computing *ACM Transactions on Quantum Computing* **3** 1–50
[108] Lubinski T, Granade C, Anderson A, Geller A, Roetteler M, Petrenko A and Heim B 2022 Advancing hybrid quantum–classical computation with real-time execution *Front. Phys.* **10** 940293
[109] Shaw A L, Finkelstein R, Tsai R B-S, Scholl P, Yoon T H, Choi J and Endres M 2024 Multi-ensemble metrology by programming local rotations with atom movements *Nat. Phys.* **20** 195–201
[110] Levine H, Bluvstein D, Keesling A, Wang T T, Ebadi S, Semeghini G, Omran A, Greiner M, Vuletić V and Lukin M D 2022 Dispersive optical systems for scalable Raman driving of hyperfine qubits *Phys. Rev.* A **105** 032618
[111] Tan D B, Bluvstein D, Lukin M D and Cong J 2024 Compiling quantum circuits for dynamically field-programmable neutral atoms array processors *Quantum* **8** 1281
[112] Wang H, Tan B, Liu P, Liu Y, Gu J, Cong J and Han S 2023 Q-Pilot: field programmable quantum array compilation with flying ancillas (arXiv:2311.16190)
[113] Litteken A 2023 Neutral atom compilation (available at: https://github.com/AndrewLitteken/neutral-atom-compilation)
[114] S4Plus 2022 Q-Tetris (available at: https://github.com/S4Plus/Q-Tetris)
[115] Patel T, Silver D and Tiwari D 2022 GEYSER (ISCA'22) code and dataset *Zenodo* https://doi.org/10.5281/zenodo.7084132
[116] UCLA VAST Lab 2023 OLSQ-DPQA compiler (available at: https://github.com/UCLA-VAST/DPQA)
[117] Chair for Design Automation - Technical University of Munich 2024 MQT QMAP (available at: https://github.com/cda-tum/mqt-qmap)
[118] Zulehner A and Wille R 2018 One-pass design of reversible circuits: combining embedding and synthesis for reversible logic *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **37** 996
[119] Soeken M, Wille R, Hilken C, Przigoda N and Drechsler R 2012 Synthesis of reversible circuits with minimal lines for large functions *17th Asia and South Pacific Design Automation Conf.* pp 85–92
[120] Wille R, Burgholzer L and Zulehner A 2019 Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations *Proc. 56th Annual Design Automation Conf. 2019 (DAC '19)* (Association for Computing Machinery) pp 1–6
[121] Biere A, Biere A, Heule M, van Maaren H and Walsh T 2009 *Handbook of Satisfiability* (IOS Press)
[122] Wagner N, Poole C, Graham T M and Saffman M 2024 Benchmarking a neutral-atom quantum computer *Int. J. Quantum Inf.* (https://doi.org/10.1142/S0219749924500011)
[123] McInroy K, Pearson N and Pritchard J D 2024 Benchmarking the algorithmic performance of near-term neutral atom processors (arXiv:2402.02127)
[124] Schmid Ludwig *et al* 2023 Dataset and evaluation scripts *Zenodo*
[125] Quetschlich N, Burgholzer L and Wille R 2023 MQT bench: benchmarking software and design automation tools for quantum computing *Quantum* **7** 1062
[126] Shende V V and Markov I L 2009 On the CNOT-cost of TOFFOLI gates *Quantum Inf. Comput.* **9** 461–86
[127] He Y, Luo M-X, Zhang E, Wang H-K and Wang X-F 2017 Decompositions of *n*-qubit Toffoli gates with linear circuit complexity *Int. J. Theor. Phys.* **56** 2350