Search Smarter, Not Harder: A Scalable, High-Quality Zoned Neutral Atom Compiler

Yannick Stade
yannick.stade@tum.de
Chair for Design Automation,
Technical University of Munich
Munich, Germany

Lukas Burgholzer
lukas.burgholzer@tum.de
Chair for Design Automation,
Technical University of Munich
Munich, Germany
Munich Quantum Software Company
Munich, Germany

Robert Wille
robert.wille@tum.de
Chair for Design Automation,
Technical University of Munich
Munich, Germany
Munich Quantum Software Company
Garching near Munich, Germany

Abstract

Zoned neutral atom architectures are emerging as a promising platform for large-scale quantum computing. Their growing scale, however, creates a critical need for efficient and automated compilation solutions. Yet, existing methods fail to scale to the thousands of qubits these devices promise. State-of-the-art compilers, in particular, suffer from immense memory requirements that limit them to small-scale problems. This work proposes a scalable compilation strategy that "searches smarter, not harder". We introduce *Iterative Diving Search* (IDS), a goal-directed search algorithm that avoids the memory issues of previous methods, and *relaxed routing*, an optimization to mitigate atom rearrangement overhead. Our evaluation confirms that this approach compiles circuits with thousands of qubits and, in addition, even reduces rearrangement overhead by 28.1% on average. The complete code is publicly available in open-source as part of the *Munich Quantum Toolkit* (MQT) at https://github.com/munich-quantum-toolkit/qmap.

CCS Concepts

• **Hardware → Quantum computation**; *Hardware description languages and compilation.*

Keywords

quantum computing, optimizing compilers, rydberg atoms, ultracold atoms, routing

1 Introduction

Quantum computing based on neutral atoms [1] is reaching a stage where efficient and scalable compilation strategies become indispensable. In particular, architectures that are split into multiple spatially separated zones featuring dedicated operations—so-called *zoned neutral atom architectures* [1, 2]—have been demonstrated as promising candidates for large-scale quantum computing with several thousand qubits [3]. These qubit counts render manual compilation strategies or adhoc Python scripts infeasible. Hence, there is a pressing need for advanced quantum compilation strategies that can efficiently transform high-level quantum algorithms into low-level instructions tailored for this specific architecture.

Unfortunately, despite initial accomplishments in compilation for zoned neutral atom architectures [4–7], existing approaches either fail to scale to the available qubit counts or compromise on result quality. In particular, the state-of-the-art compiler [6] utilizing *routing-aware placement* improves significantly over previous approaches, e. g., ZAC [5]. However, due to immense memory usage, it struggles to compile circuits with more than a few hundreds of qubits and more than a few dozens of parallel entangling gates. Hence, the question remains how to develop a scalable *and* high-quality compilation strategy.

Instead of searching *harder* by using tens of gigabytes, this work proposes an alternative heuristic compilation strategy. It combines the established principles of A* search with a much *smarter* exploration strategy. To this end, we analyze the key aspects in existing compilers allowing them to achieve high-quality results, i. e., low rearrangement overhead. Based on this analysis, we combine these aspects with a novel search algorithm, called *Iterative Diving Search* (IDS), that overcomes the scalability issues in previous work—unlocking compilation for large-scale architectures. To mitigate the increasing rearrangement overhead in large-scale architectures, we propose *relaxed routing*, an optimization that leverages a common conflict-avoidance strategy reducing rearrangement overhead specifically for long distances. Overall, this eventually leads to a solution that easily scales to thousands of qubits and hundreds of parallel entangling gates while even improving result quality.

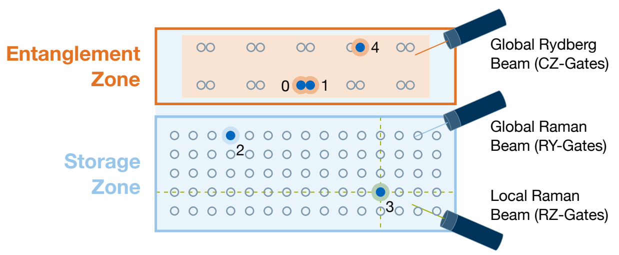
Evaluations show that the resulting approach allows, for the first time, to consider instances with up to 5000 qubits and 300 parallel entangling gates while maintaining high compilation quality. It compiles all considered circuits within minutes, while the state-of-the-art approach frequently runs into memory outs. Moreover, it even further improves the compilation quality, i. e., the rearrangement overhead by 27.1% on average compared to the state-of-the-art method. Furthermore, the evaluations also unveil an effect of the relaxed routing optimization yielding an overall improvement of 28.1% on average compared to the state-of-the-art. The implementation of the proposed approach is publicly available as part of the *Munich Quantum Toolkit* (MQT, [8]) at https://github.com/munich-quantum-toolkit/qmap.

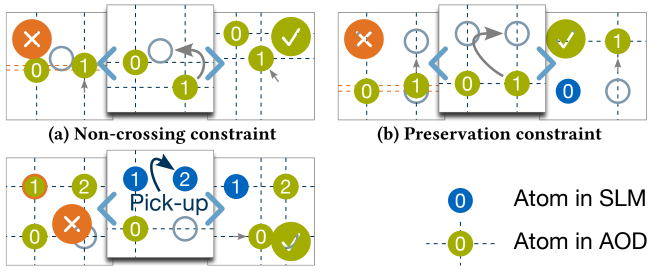
2 Background

To provide the necessary context for the compilation problem in zoned neutral atom architectures, this section reviews their computational capabilities and corresponding compilation strategies.

2.1 Zoned Neutral Atom Architectures

In quantum computing based on neutral atoms, qubits are encoded in the electronic states of individual neutral atoms from the group of alkali or alkaline-earth metals. The atoms are trapped by optical


Entanglement Zone
Storage Zone
Global Rydberg Beam (CZ-Gates)
Global Raman Beam (RY-Gates)
Local Raman Beam (RZ-Gates)
0
1
2
3
4
Figure 1: Schematic drawing of a zoned neutral atom architecture featuring an entanglement (orange) and a storage zone (light blue). Local and global laser beams execute gates.



(a) Non-crossing constraint **(b) Preservation constraint**

(c) Ghost-spot constraint

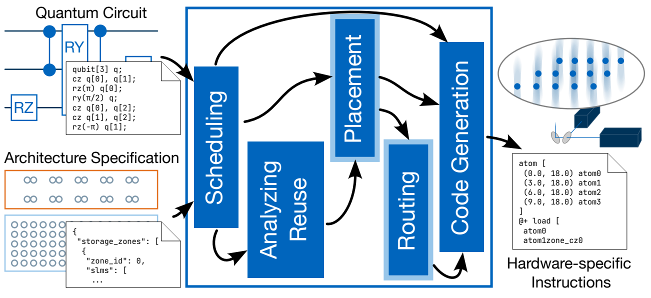
Figure 2: The middle frame of each sub-figure shows the intended movement, the left one a violation, and the right one a possible workaround.

tweezers or lattices [9]. Architectures typically feature one or multiple rectangular grids of such optical traps representing potential atom locations [2] depicted as gray circles in Fig. 1.

Based on that, operations are realized as follows: single-qubit operations, such as rotations on these atoms, are applied through laser-driven electronic state transitions [10]. Two-qubit operations such as the controlled phase gate (CZ gate), the common native entangling operation on neutral atom architectures, are realized by the *Rydberg blockade* mechanism [2, 11]. When illuminated with the so-called Rydberg beam, it ensures that only qubits within a certain *interaction radius* of each other interact, while all illuminated atoms, also isolated ones, experience a certain likelihood of an error.

These operations can either be applied locally on individual atoms or globally on all atoms within a defined extent, referred to as a *zone* [2]. In particular, when illuminating the entire entanglement zone with a Rydberg laser, all atoms within that zone are excited to the Rydberg state, but only those pairs of atoms (atom 0 and 1 in Fig. 1) that are close enough undergo a CZ gate; the others (atom 4) undergo an error-prone identity operation. To improve the overall fidelity of the computation, zoned architectures spatially separate operations into three different kinds of zones: entanglement zones for two-qubit operations, storage zones featuring long coherence times, and measurement zones for readout [1, 2].

The execution of a quantum circuit on a zoned neutral atom architecture involves the following steps: (1) perform all currently executable single-qubit operations, (2) move pairs of atoms that should interact to the entanglement zone, (3) perform all currently executable two-qubit operations, (4) move not needed atoms back to the storage zone, (5) if applicable, move atoms to the measurement zone for measurements, and (6) repeat until the entire circuit is executed. The movements are achieved by two types of traps: static



```
qubit[3] q;
cz q[0], q[1];
rz(π) q[0];
ry(π/2) q;
cz q[0], q[2];
cz q[1], q[2];
rz(-π) q[1];
```

```
{
 "storage_zones": [
  {
   "zone_id": 0,
   "slms": [
    ...
```

```
atom [
 (0.0, 18.0) atom0
 (3.0, 18.0) atom1
 (6.0, 18.0) atom2
 (9.0, 18.0) atom3
]
@+ load [
 atom0
 atom1zone_cz0
```

Figure 3: Compilation flow from the quantum circuit and architecture specification to hardware-specific instructions.

Spatial Light Modulator (SLM) traps that hold the atoms in place and adjustable *Acousto-Optic Deflector* (AOD) traps. Two orthogonal AODs create a rectangular grid of adjustable traps where each row and column of the grid can be controlled individually. To move an atom, its corresponding column and row are activated to pick it up, shifted to the new position, and then deactivated to drop it off [10]. While multiple atoms can be moved in parallel, the following *rearrangement constraints* must be followed [4]:

(1) Columns/rows must maintain a minimal separation; in particular, they cannot cross (cf. Fig. 2a).
(2) Activated columns/rows cannot split or merge (cf. Fig. 2b).
(3) Unwanted, so-called *ghost-spots*, must be avoided (cf. Fig. 2c).

One so-called *rearrangement step* starts with picking up some atoms and lasts until dropping off the last atoms.

2.2 Related Work

Compilation for various neutral atom architectures has been an active research area in recent years. Some of them employ a heuristic approach [12–22], while others use exact methods [23–25].

For zoned neutral atom architectures in particular, the following compilation flow as illustrated in Fig. 3 has been established. It typically consists of the following five steps [4–7]:

(1) *Scheduling*: divides the circuit into layers of single- and parallel executable two-qubit operations.
(2) *Reuse Analysis*, proposed in [5], reduces rearrangement overhead by identifying atoms that can remain in the entanglement zone across multiple layers.
(3) *Placement* assigns atoms, for each layer, to locations.
(4) *Routing* groups movements into rearrangement steps.
(5) *Code Generation* produces hardware-specific instructions.

The first compiler for zoned neutral atom architectures was NALAC [4]. This compiler, however, also excites idling atoms in the entanglement zone leading to avoidable errors. To this end, ZAC [5] improves upon NALAC by moving back all atoms to the storage zone when not needed anymore. Additionally, it introduces *reuse-aware* compilation to further improve the overall quantum circuit fidelity. However, ZAC employs a fast but suboptimal placement strategy based on minimum weight perfect matching. It purely focuses on minimizing the atoms' travel distances, leading to many conflicts with the rearrangement constraints (cf. Sec. 2.1) in the subsequent routing stage. Due to these conflicts, many movements must be serialized causing many rearrangement steps and resulting in high rearrangement overhead. PowerMove [7] follows a similar approach with the same drawbacks. These shortcomings are addressed by the *routing-aware placement* [6] constituting the current state-of-the-art in compilation for zoned neutral atom architectures. By considering the rearrangement constraints already during placement, it significantly reduces the number of required rearrangement steps. Despite this improvement, the employed A* algorithm suffers from severe scalability issues. In particular, the number of parallel entangling gates in the circuit is crucial for the memory consumption of this compilation strategy. Compiling circuits with just a few dozen parallel CZ gates per layer can demand over 40 GB of memory, rendering it impractical for larger instances.

3 Scalable Compilation: Search Smarter, Not Harder

This work introduces a compilation strategy that "searches smarter, not harder" to overcome the aforementioned limitations. To this end, we first identify the key features of existing methods responsible for high-quality results and then incorporate them into a novel, more goal-directed search algorithm. This approach aims to preserve, and even enhance, the result quality while ensuring scalability to thousands of qubits and hundreds of parallel entangling gates.

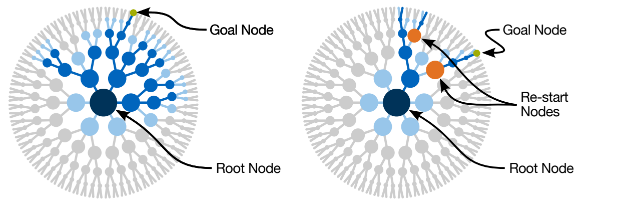
3.1 Status Quo: What Makes High-Quality Compilation?

To mitigate errors caused by decoherence, minimizing the overall computation time of a quantum computation is crucial. On zoned neutral atom architectures, this translates directly into minimizing the *rearrangement overhead*, i. e., the time spent rearranging atoms between layers. Since the number of atoms that must be rearranged is given by the circuit's structure, the key determinant to minimize this overhead is the number of atoms that can be rearranged in parallel in a single rearrangement step. All the individual movements in one rearrangement step must comply with the rearrangement constraints outlined in Sec. 2.1. Hence, low rearrangement overhead can be achieved by choosing clever placements of atoms for each layer that facilitate rearrangements with many parallel movements.

To this end, the placement stage in Fig. 3 can be modeled as a tree search problem where each node represents a (partial) placement of atoms. The root node represents the initial placement before any atoms have been placed for the current layer. To simplify the process, only atoms that need to move, i. e., to or from the entanglement zone are considered. Each child node extends the placement of its parent node by placing one more additional atom. A node is a goal node if all atoms have been placed.

To differentiate between good and bad placements, a cost and heuristic function is defined. The cost function assigns a cost to each node proportional to the rearrangement overhead incurred by the partial placement represented by that node. The heuristic estimates the remaining cost to reach a goal node, i. e., a complete placement, from the current node. These functions can be used by heuristic search algorithms, such as the A* algorithm [26].

The state-of-the-art method demonstrates that a well-designed heuristic and cost function can effectively identify placements that minimize rearrangement overhead [6]. Unlike previous solutions, this method prioritizes placements that preserve the atoms' relative positioning during rearrangement, even at the cost of slightly



(a) Despite the focus towards goal nodes, the A* algorithm explores the tree more in breath and visits a lot of nodes.

(b) The depth-first search-based IDS algorithm explores the tree more streered towards the goal nodes, resulting in fewer nodes being visited.

Figure 4: When searching for a path from the root node (middle) to a good goal node (green), both algorithms explore nodes (highlighted in blue; the expanded ones in dark blue).

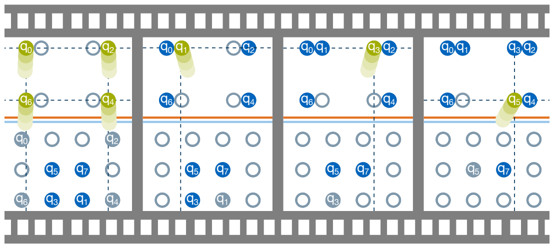
longer travel distances. This approach reduces conflicts with rearrangement constraints, enabling more parallel movements per step. Consequently, a key factor for a successful compilation strategy is a heuristic and cost function that successfully encodes the rearrangement constraints to favor placements that maximize parallelism. However, for an efficient solution, these functions must be combined with a scalable search strategy.

Example 3.1. The state-of-the-art routing-aware placement [6] employs the A* algorithm [26] to search for a low-cost goal node in the search tree. This leads to a graph exploration as schematically illustrated in Fig. 4a starting from the middle node marked in green and ending at a (solid green) goal node on the outer circle. As the illustration shows, the A* algorithm explores a lot of nodes in breath—perfect for an exhaustive search potentially yielding a high-quality result. However, this also leads to immense memory consumption as all these nodes need to be stored in memory for the A* algorithm. By that, the search becomes harder and harder as the search space grows, ultimately limiting this approach to only a few dozens of parallel CZ gates per layer in practice.

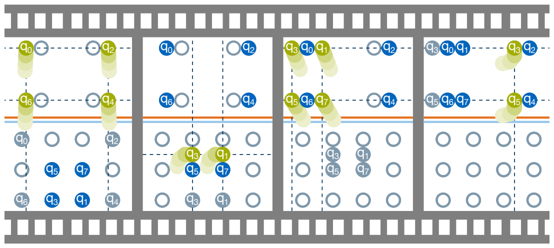
3.2 Quo Vadis? A More Goal-Directed Search

The state-of-the-art compilation strategy quickly requires more than 40 GB of memory when compiling layers with more than a few dozens parallel CZ gates, rendering it impractical for larger instances. To overcome the memory issue, we propose a *smarter* search strategy that avoids the *hard* job of a very exhaustive search while still maintaining high-quality results. To this end, we propose *Iterative Diving Search* (IDS)—a novel search strategy still based on the A* algorithm but exploring the search tree in a more depth-first manner as illustrated in Fig. 4b.

Example 3.2. As in Ex. 3.1, IDS also starts from the initial node in the middle marked green but then dives straight down the tree towards a goal node or a leaf (i. e., node without children). This behavior resembles a *Depth-First Search* (DFS) with the important aspect that the most promising child node, according to the heuristic, is always selected for expansion. After reaching a goal node or a leaf, instead of backtracking like DFS, it jumps to the most promising node discovered so far and restarts the search from there. In Fig. 4b, the search is restarted twice, indicated by the nodes marked in orange. This process is repeated until a constant, configurable number of goal nodes have been found and the best goal node discovered so far is returned.



(a) Strict routing requires five rearrangment steps (the last step is omitted for brevity), as described in Ex. 3.3.



(b) Relaxed routing accomplishes the entire rearrangement in only two steps by shifting the lower row upwards before picking up the next row and allowing the left column to shift right after dropping off the right column.

Figure 5: Comparison of routing approaches for an interaction of the atom pairs $(0, 1)$, $(2, 3)$, $(4, 5)$ and $(6, 7)$.

This more effective search strategy is combined with the cost and heuristic functions that already have proven successful in previous work [6] but could easily be updated with upcoming functions.

3.3 Final Touches: Making a Virtue out of Necessity

Picking up atoms from multiple rows and columns simultaneously can create unwanted ghost-spots, which inadvertently trap atoms that should remain in place. A common strategy in existing compilers to avoid these ghost-spots is to pick up the atoms row-by-row with a small offset move in between [5, 6]. Still, they leave the order of rows and columns unchanged during one rearrangement step.

Example 3.3. In Fig. 5, a CZ gate is to be performed between the atom pairs $(0, 1)$, $(2, 3)$, $(4, 5)$, and $(6, 7)$. To this end, the atoms need to be aligned accordingly in the entanglement zone (above the orange line). Using strict routing, four atoms can be moved in one rearrangement step as shown in the first frame of Fig. 5a. However, to avoid ghost-spots, strict routing requires four more separate rearrangement steps to move the remaining atoms.

One can make a virtue out of this necessity to avoid ghost-spots by dropping the *strict* constraint that the rows must remain in order during a rearrangement step. This constraint can be *relaxed* by allowing already activated rows to shift vertically before the next row is picked up. Analogously, when allowing to drop off atoms column-by-column (instead of row-by-row) and shifting still activated columns horizontally after dropping off a column, also the columns can be reordered. We call this new approach *relaxed routing* in contrast to the established *strict routing*.

Example 3.4. Relaxed routing, as shown in Fig. 5b, picks up atom 1 and 3 (2nd frame), shifts them upwards, and then picks up atom 5 and 7 (3rd frame)—effectively reordering the rows. Similarly, after dropping off atom 1 and 7 (4th frame), the left column is shifted right after the first drop-off—effectively reordering the columns.

4 Implementation

The realization of the proposed methods requires careful implementation. The following sections outline the details on the implementation of IDS and the tuning of relaxed routing.

4.1 Iterative Diving Search

The following pseudocode implements the proposed *Iterative Diving Search* (IDS). Without the parts highlighted in blue, the pseudocode implements the classical A* algorithm [26].[1]

```
 1: function IDS(tree $\mathcal{T}$, vertex $v_{\text{start}}$, $n_{\text{trials}} > 0$)
 2:     $Q \leftarrow$ min-priority queue with max. $N_{\max}$ elements
 3:     $v \leftarrow v_{\text{start}}$
 4:     loop
 5:         if $v$ is goal then
 6:             if $v_{\text{goal}}$ is NIL or cost($v$) < cost($v_{\text{goal}}$) then
 7:                 $v_{\text{goal}} \leftarrow v$
 8:             $n_{\text{trials}} \leftarrow n_{\text{trials}} - 1$
 9:             if $n_{\text{trials}} = 0$ then
10:                 break
11:         else
12:             $M \leftarrow$ children $u$ of $v$ in $\mathcal{T}$
13:             if $M \neq \emptyset$ then
14:                 $v \leftarrow u \in M$ with min. cost
15:                 for $u \in S \setminus \{v\}$ do
16:                     $Q$.push($u$, cost($u$))                ▷ 2nd arg. is prio.
17:                 continue
18:         if $Q$ empty then
19:             break
20:         $v \leftarrow Q$.pop()
21:     return $v_{\text{goal}}$
```

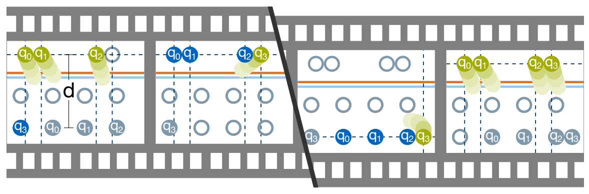
IDS uses a min-priority queue Q to keep track of the most promising nodes encountered so far that have not yet been expanded (line 2). To bound the memory consumption, the maximum number of elements $N_{\max}$ in Q is limited. This may cause some nodes to be discarded when Q is full, leading to a non-exhaustive search. When a new goal node is found, the best goal node discovered so far is updated (lines 6–7). Additionally, the variable n_{trials}—used to keep track of how many goal nodes should be found before terminating the search—is decremented accordingly (lines 8–9).

While expanding a node with children, the child with the lowest cost is *not* added to Q and directly selected for expansion (lines 13–14). This implements the depth-first exploration of IDS. If the node is a leaf, the most promising node from Q is selected for expansion (lines 18–19)—the same applies after a goal node has been discovered. The search either ends when n_{trials} reaches zero or when no more nodes are available for expansion (lines 10 and 19).

4.2 Tuning Relaxed Routing

It is expected that the relaxed routing approach especially pays off for long rearrangement distances. This is due to the effect that the

[1]Note that the search space resembles a tree, and hence there is no need to remember visited nodes because there are no cycles.



(a) Strict Routing (b) Relaxed Routing

Figure 6: (a) To let the atom pairs $(0, 1)$ and $(2, 3)$ interact, the strict routing approach requires two separate rearrangement steps. (b) By first picking up atom 3, moving it right of atom 2, and then picking up the remaining atoms, the relaxed routing can achieve the same rearrangement in one go. The cost for the additional offset move particularly pays off when the distance d indicated in Fig. 6a is large.

additional offset moves introduced by relaxed routing also incur some overhead. For short rearrangements, this additional overhead might even outweigh the benefits of relaxed routing.

Example 4.1. Figure 6 shows the rearrangements to execute CZ gates between the atom pairs $(0, 1)$ and $(2, 3)$ using strict routing in Fig. 6a, and relaxed routing in Fig. 6b. When the distance d is small, the offset move of atom 3 in Fig. 6b incurs a higher time overhead than the extra rearrangement step required by strict routing in Fig. 6a. However, as d increases, the time overhead of the extra rearrangement step outweighs the overhead of the offset movement.

Hence, there exists a crossover point depending on the rearrangement distance where relaxed routing starts to pay off. To this end, we estimate the additional overhead caused by relaxed routing and fall back to strict routing if preferable.

5 Evaluation

The proposed strategy is the first one that combines high-quality results and scalability. To evaluate the effectiveness of the proposed methods, we compare its performance against the state-of-the-art compilation strategy for zoned neutral atom architectures [6]. This section starts with a review of the experimental setup and a parameter study, followed by a presentation and discussion of the obtained results. The complete code is publicly available in open-source as part of the MQT [8] under https://github.com/cda-tum/mqt-qmap/.

5.1 Experimental Setup and Parameter Study

We implemented the proposed method in C++ on top of the existing state-of-the-art compiler [6]. All experiments were conducted on an Apple M3 with 16GB of RAM. During compilation, a memory limit of 20 GB was enforced for each instance.

We considered the zoned neutral atom architecture proposed in [2] slightly enlarged to an overall dimension of 400 by 400 µm to fit the benchmarks. The architecture is composed of two zones: a storage zone with a 73×101 grid of traps separated by 4 µm, and an entanglement zone with a 10×34 grid of trap pairs. Each pair has two traps positioned (horizontally) 2 µm apart, and the pairs are spaced to maintain a minimum distance of 10 µm between traps from adjacent pairs. The zones are vertically arranged, horizontally centered, and have a minimum separation of 21 µm.

To benchmark the proposed approach, we have selected various quantum circuits of varying sizes from MQT Bench [27], as listed

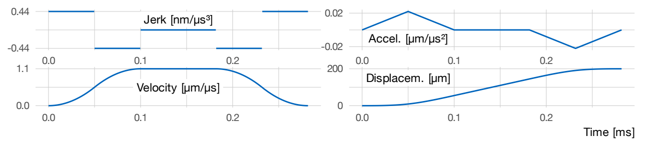


Figure 7: Movement profile with a constant jerk and a maximum speed.

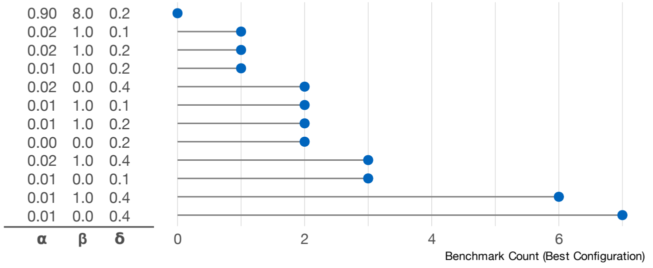


Figure 8: The plot counts the benchmarks (x-axis) for which a specific configuration (y-axis) yielded the best result.

in the first two columns of Table 1. The next four columns list key characteristics of the circuits; in particular, the maximum number of parallel two-qubit gates per layer being the most critical factor for the size of the search space, as discussed in Sec. 2.2. Due to their structure, most of these circuits yield a low number of parallel entangling gates per layer. One exception is the `graphstate` family, where the degree of parallelism grows with the circuit size until the capacity of the architecture's entanglement zone is reached.[2] We consider these highly parallel circuits representative of circuits specifically tailored for neutral atom architectures, which offer more parallelism than, for instance, superconducting architectures.

As the central metric, to evaluate the compilation quality, we measure the time overhead caused by rearrangements during the execution of the compiled circuit, referred to as *rearrangement time*. This metric directly impacts the overall fidelity of the computation because it contributes to the total computation time during which decoherence can occur. We assume that pick-up and drop-off operations take 15 µs [2, 5] and employ a movement profile based on a constant jerk of 0.44 $\frac{\text{nm}}{\text{s}^3}$ with a maximum speed of 1.1 $\frac{\text{µm}}{\text{µs}}$ [2]. Figure 7 shows the resulting movement profile for a distance of 200 µm. The resulting time for this movement is 0.24 ms.

As explained in Sec. 3.2, the state-of-the-art and the proposed method employ the same tunable heuristic function. For the state-of-the-art approach, we use the best parameter configuration identified for large circuits in [6]. However, since our experiments indicated this configuration is suboptimal for the proposed method, we conducted a parameter study to find a better-suited configuration for the proposed approach. To this end, we varied the parameters δ, β (accelerating part of the heuristic), and α (look-ahead, cf. [6]) and summarized the results in Fig. 8. The best results were achieved with the configuration $\delta = 0.01$, $\beta = 0.0$, and $\alpha = 0.4$.

5.2 Obtained Results

Using the setup and the best parameter configuration identified in Sec. 5.1, we eventually compare the proposed method against the

[2]The architecure's limit is 306 parallel CZ gates as the compiler employs a 90% filling at maximum.

Table 1: Results of Iterative Diving Search

Benchmark				Max.	**A*** *(State-of-the-Art)*		**IDS** *(Proposed Solution)*		**Relaxed Routing** *(Proposed Opt.)*
	Num. Qubits	Num. 2Q-Gates	Num. Layers	2Q-Gates in Layer	Comp. Time [s]	Rearr. T. [ms]	Comp. Time [s]	Rearr. Time [ms]	Rearr. Time [ms]
qft	500	18 620	1994	10	4.0	2786.3	4.2 (+0.2)	2324.5 (−16.6%)	2287.1 (−17.9%)
	1000	37 620	3994	10	4.9	5816.2	6.2 (+1.3)	4937.9 (−15.1%)	4855.8 (−16.5%)
wstate	500	998	501	2	0.5	421.1	0.6 (+0.1)	410.8 (−2.4%)	410.5 (−2.5%)
	1000	1998	1001	2	1.0	945.0	1.0 (±0.0)	777.7 (−17.7%)	777.1 (−17.8%)
qpeexact	500	19 579	2987	10	5.3	3316.4	5.3 (±0.0)	2867.5 (−13.5%)	2825.2 (−14.8%)
	1000	39 579	5987	10	MEMOUT		6.7 (NA)	5928.0 (NA)	5845.1 (NA)
vqe two local	50	3675	197	25	2.7	806.5	2.7 (−0.1)	631.9 (−21.7%)	624.7 (−22.5%)
	100	14 850	397	50	20.5	3662.5	14.7 (−5.8)	2423.1 (−33.8%)	2394.6 (−34.6%)
	150	33 525	597	75	63.0	7994.3	47.5 (−15.6)	4432.0 (−44.6%)	4374.9 (−45.3%)
	200	59 700	797	100	MEMOUT		120.1 (NA)	8477.0 (NA)	8349.7 (NA)
qaoa	50	2348	248	16	1.3	410.0	1.4 (±0.0)	270.1 (−34.1%)	267.5 (−34.8%)
	100	9736	508	31	MEMOUT		8.2 (NA)	995.2 (NA)	979.8 (NA)
	150	22 256	762	47	MEMOUT		26.7 (NA)	2026.5 (NA)	1996.7 (NA)
	200	39 552	1028	61	MEMOUT		67.6 (NA)	3650.8 (NA)	3593.9 (NA)
graphstate	60	60	6	22	0.1	16.6	0.1 (±0.0)	14.8 (−11.2%)	14.8 (−11.2%)
	80	80	6	30	0.2	31.9	0.2 (+0.1)	19.7 (−38.1%)	19.4 (−39.3%)
	100	100	10	34	0.3	40.8	0.3 (±0.0)	22.4 (−45.1%)	22.1 (−45.9%)
	120	120	6	43	10.5	45.0	0.5 (−10.0)	28.7 (−36.3%)	28.4 (−36.8%)
	140	140	9	48	MEMOUT		0.5 (NA)	36.6 (NA)	36.2 (NA)
	160	160	6	59	MEMOUT		1.5 (NA)	38.0 (NA)	37.7 (NA)
	200	200	7	77	MEMOUT		1.2 (NA)	53.0 (NA)	51.8 (NA)
	500	500	6	184	MEMOUT		8.0 (NA)	133.8 (NA)	132.8 (NA)
	1000	1000	7	306	MEMOUT		57.6 (NA)	320.5 (NA)	320.9 (NA)
	2000	2000	9	306	MEMOUT		227.4 (NA)	802.7 (NA)	802.1 (NA)
	5000	5000	18	306	MEMOUT		647.7 (NA)	2588.8 (NA)	2582.3 (NA)
∅					8.8	2022.5	50.3 (−2.3)	1768.5 (−27.1%)	1745.2 (−28.1%)

state-of-the-art method proposed in [6]. Table 1 summarizes the obtained results across all benchmarks. The middle two regions of Table 1 list the compilation times and resulting rearrangement times for the state-of-the-art A*-based method and the proposed IDS method, respectively. The final column shows rearrangement times for IDS combined with relaxed routing.

The results reveal that compilation times of both methods are comparable with the notable exception that the proposed method requires significantly less memory and never hits the memory limit of 20 GB. This allows the proposed method to successfully compile larger instances while the state-of-the-art method frequently fails due to memory outs. Comparing the resulting rearrangement times as the most critical quality metric, the proposed method consistently outperforms the state-of-the-art method, achieving an average improvement of 27.1% and up to 45.1% on highly parallel benchmarks like the `graphstate` family. Adding the proposed relaxed routing further reduces rearrangement overhead—reaching an overall improvement of up to 45.9% for specific benchmarks and 28.1% on average compared to the previous state-of-the-art.

These results demonstrate that the proposed method's more goal-directed search strategy finds significantly better placements despite being less exhaustive. Unlike the A*-based approach, which requires a heuristic balancing solution quality with search acceleration [6], the inherent efficiency of the proposed method allows for a heuristic focused solely on solution quality, resulting in superior placements. Moreover, relaxed routing can successfully mitigate rearrangement overhead that grows with circuit size.

6 Conclusions

This work addressed the scalability limitations of existing compilers for zoned neutral atom architectures. We introduced a scalable compilation strategy, called *Iterative Diving Search* (IDS), and a novel optimization called *relaxed routing*. Evaluations show that this approach not only scales to thousands of qubits but also reduces the rearrangement overhead by 28.1% on average compared to the state-of-the-art. These advancements enable efficient compilation of large-scale, highly parallel quantum circuits on near-term neutral atom quantum computers.

Acknowledgments

The authors acknowledge funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 101001318), and the Munich Quantum Valley (MQV), which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus.

References

[1] Dolev Bluvstein et al. *Architectural Mechanisms of a Universal Fault-Tolerant Quantum Computer*. 2025. arXiv: 2506.20661. Pre-published.

[2] Dolev Bluvstein et al. “Logical Quantum Processor Based on Reconfigurable Atom Arrays”. In: *Nature* (2023). DOI: 10.1038/s41586-023-06927-3.

[3] Neng-Chun Chiu et al. *Continuous Operation of a Coherent 3,000-Qubit System*. 2025. arXiv: 2506.20660. Pre-published.

[4] Yannick Stade et al. “An Abstract Model and Efficient Routing for Logical Entangling Gates on Zoned Neutral Atom Architectures”. In: *Int’l Conf. on Quantum Computing and Engineering*. IEEE, 2024. DOI: 10.1109/QCE60285.2024.00098.

[5] Wan-Hsuan Lin, Daniel Bochen Tan, and Jason Cong. “Reuse-Aware Compilation for Zoned Quantum Architectures Based on Neutral Atoms”. In: *IEEE Int’l Symp. on High-Performance Computer Architecture*. 2025. arXiv: 2411.11784.

[6] Yannick Stade et al. “Routing-Aware Placement for Zoned Neutral Atom-based Quantum Computing”. In: 2025. arXiv: 2505.22715.

[7] Jixuan Ruan et al. *PowerMove: Optimizing Compilation for Neutral Atom Quantum Computers with Zoned Architecture*. 2024. DOI: 10.48550/ARXIV.2411.12263.

[8] Robert Wille et al. “The MQT Handbook: A Summary of Design Automation Tools and Software for Quantum Computing”. In: *Int’l Conf. on Quantum Software*. 2024. arXiv: 2405.17543. A live version of this document is available at https://mqt.readthedocs.io.

[9] Daniel Barredo et al. “An Atom-by-Atom Assembler of Defect-Free Arbitrary Two-Dimensional Atomic Arrays”. In: *Science* (2016). DOI: 10.1126/science.aah3778.

[10] Dolev Bluvstein et al. “A Quantum Processor Based on Coherent Transport of Entangled Atom Arrays”. In: *Nature* (2022). DOI: 10.1038/s41586-022-04592-6.

[11] Simon J. Evered et al. “High-Fidelity Parallel Entangling Gates on a Neutral Atom Quantum Computer”. In: *Nature* (2023). DOI: 10.1038/s41586-023-06481-y.

[12] Jonathan M. Baker et al. “Exploiting Long-Distance Interactions and Tolerating Atom Loss in Neutral Atom Quantum Architectures”. In: *Int’l Symposium on Computer Architecture*. IEEE, 2021. DOI: 10.1109/ISCA52012.2021.00069.

[13] Tirthak Patel, Daniel Silver, and Devesh Tiwari. “Geyser: A Compilation Framework for Quantum Computing with Neutral Atoms”. In: *Int’l Symposium on Computer Architecture*. ACM, 2022. DOI: 10.1145/3470496.3527428.

[14] Natalia Nottingham et al. “Decomposing and Routing Quantum Circuits Under Constraints for Neutral Atom Architectures”. In: *Int’l Conf. on Quantum Computing and Engineering*. 2024. arXiv: 2307.14996.

[15] Tirthak Patel, Daniel Silver, and Devesh Tiwari. “GRAPHINE: Enhanced Neutral Atom Quantum Computing Using Application-Specific Rydberg Atom Arrangement”. In: *Int’l Conf. for High Performance Computing, Networking, Storage and Analysis*. ACM, 2023. DOI: 10.1145/3581784.3607032.

[16] Ludwig Schmid, Sunghye Park, and Robert Wille. “Hybrid Circuit Mapping: Leveraging the Full Spectrum of Computational Capabilities of Neutral Atom Quantum Computers”. In: *Design Automation Conf.* ACM, 2024. DOI: 10.1145/3649329.3655959.

[17] Hanrui Wang et al. “Atomique: A Quantum Compiler for Reconfigurable Neutral Atom Arrays”. In: *Int’l Symposium on Computer Architecture*. IEEE, 2024. DOI: 10.1109/ISCA59077.2024.00030.

[18] Yunqi Huang et al. *DasAtom: A Divide-and-Shuttle Atom Approach to Quantum Circuit Transformation*. 2024. arXiv: 2409.03185.

[19] Daniel Silver, Tirthak Patel, and Devesh Tiwari. *Qompose: A Technique to Select Optimal Algorithm- Specific Layout for Neutral Atom Quantum Architectures*. 2024. arXiv: 2409.19820.

[20] Hanrui Wang et al. “Q-Pilot: Field Programmable Qubit Array Compilation with Flying Ancillas”. In: *Design Automation Conf.* ACM, 2024. DOI: 10.1145/3649329.3658470.

[21] Jason Ludmir and Tirthak Patel. “PARALLAX: A Compiler for Neutral Atom Quantum Computers under Hardware Constraints”. In: *Int’l Conf. for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2024. DOI: 10.1109/SC41406.2024.00079.

[22] Daniel Bochen Tan, Wan-Hsuan Lin, and Jason Cong. “Compilation for Dynamically Field-Programmable Qubit Arrays with Efficient and Provably Near-Optimal Scheduling”. In: *Asia and South Pacific Design Automation Conf.* ACM, 2025. DOI: 10.1145/3658617.3697778.

[23] Nathan Constantinides et al. *Optimal Routing Protocols for Reconfigurable Atom Arrays*. 2024. arXiv: 2411.05061.

[24] Daniel Bochen Tan et al. “Compiling Quantum Circuits for Dynamically Field-Programmable Neutral Atoms Array Processors”. In: *Quantum* (2024). DOI: 10.22331/q-2024-03-14-1281.

[25] Yannick Stade et al. “Optimal State Preparation for Logical Arrays on Zoned Neutral Atom Quantum Computers”. In: *Design, Automation and Test in Europe*. 2024. DOI: 10.23919/DATE64628.2025.10993241.

[26] Peter Hart, Nils Nilsson, and Bertram Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Trans. on Systems, Man, and Cybernetics* (1968). DOI: 10.1109/TSSC.1968.300136.

[27] Nils Quetschlich, Lukas Burgholzer, and Robert Wille. “MQT Bench: Benchmarking Software and Design Automation Tools for Quantum Computing”. In: *Quantum* (2023). MQT Bench is available at https://www.cda.cit.tum.de/mqtbench/. DOI: 10.22331/q-2023-07-20-1062.