Q-Pilot: Field Programmable Qubit Array Compilation with Flying Ancillas

Hanrui Wang[1*], Daniel Bochen Tan[2*], Pengyu Liu[3], Yilian Liu[4], Jiaqi Gu[5], Jason Cong[2], Song Han[1]
[1]MIT, [2]University of California, Los Angeles, [3]Carnegie Mellon University, [4]Cornell University, [5]Arizona State University, *Equal Contributions

ABSTRACT

Neutral atom arrays, particularly the reconfigurable field programmable qubit arrays (FPQA) with atom movement, show strong promise for quantum computing. FPQA has a dynamic qubit connectivity, facilitating cost-effective execution of long-range gates, but it also poses new challenges in the compilation. Inspired by the FPGA compilation strategy, we develop a router, Q-Pilot, that leverages flying ancillas to implement 2-Q gates between data qubits mapped to fixed atoms. Equipped with domain-specific routing techniques, Q-Pilot achieves 1.4×, 27.7×, and 6.7× reductions in circuit depth for 100-qubit random, quantum simulation, and QAOA circuits, respectively, compared to alternative fixed atom array architectures.

1 INTRODUCTION

Quantum computing (QC) hardware has seen rapid scaling, with superconducting systems offering up to 433 qubits [1–4], and neutral atom arrays reaching 1000+ qubits [5, 25]. Utilizing these machines requires mapping qubits in a quantum program/circuit to physical qubits on the QPU, typically constrained by limited connectivity given by a coupling graph. For example, Fig. 1a illustrates a simple QPU with four physical qubits connected in a ring. 2-Q entangling gates, crucial for quantum programs, are restricted to adjacent physical qubits (e.g., (p_0, p_1)). Consider a quantum program with gates $\texttt{CZ}(q_0, q_1)$, $\texttt{CZ}(q_1, q_2)$, and $\texttt{CZ}(q_2, q_0)$. In Fig. 1b, the initial qubit mapping is $q_i \mapsto p_i$ for $i = 0, 1, 2$. While this mapping supports the first two gates, $\texttt{CZ}(q_2, q_0)$ involves non-adjacent p_2 and p_0. Here, a SWAP gate is inserted to *route* qubits, transforming the mapping. However, SWAP is costly: it can increase circuit depth, leading to more decoherence noise, and typically requires three 2-Q entangling gates, accumulating gate errors. Given the current QPUs' relatively high noise levels, as quantum circuits grow, it becomes crucial that compilers minimize the overheads incurred by mapping and routing to optimize performance [7, 8, 14, 18, 22–24, 26, 29–31, 34–37].

A recent breakthrough enables atom movement during quantum circuit execution [10], profoundly impacting compilation by introducing dynamic coupling graphs for QPUs, as opposed to static configurations (Fig.1). In this work, we focus on a field programmable qubit array (FPQA) architecture that incorporates this technology. FPQA features two atom types (Fig.2): SLM atoms (blue) are *fixed* atoms in traps generated by a spatial light modulator (SLM); AOD atoms (yellow) are *movable* atoms in traps generated by a 2D acousto-optic deflector (AOD). The 2D AOD, a product of two 1D AODs, allows us to specify X coordinates for columns (yellow dashes) and Y coordinates for rows. Consequently, AOD qubits move *by entire rows and columns*. To avoid non-deterministic behavior from trap overlap, we *prohibit AOD rows/columns from moving over others*. Physically, atom movement is a high-fidelity operation primarily constrained by coherence time: with only 0.1% coherence time, an atom can traverse

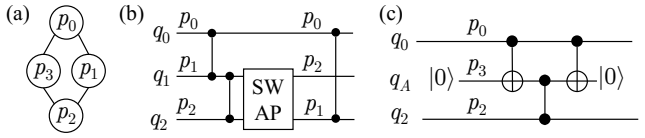

Figure 1: (a) The coupling graph of a QPU. (b) Qubit mapping and routing. The initial mapping is annotated at the beginning of each wire/qubit. A SWAP gate changes the mapping. (c) Using an ancilla and two more CNOTs to implement $\texttt{CZ}(q_0, q_2)$.

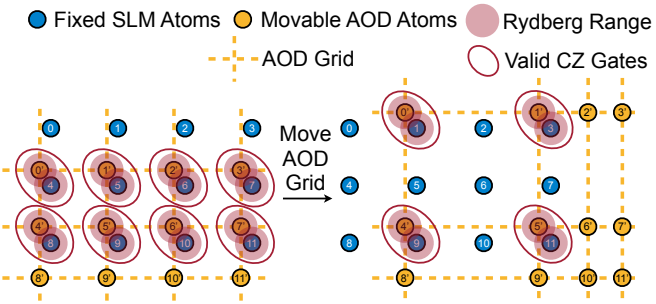

Figure 2: Field Programmable Qubit Array (FPQA).

a region for ∼ 2,000 qubits [10]. These movements are explicitly applied for 2-Q gates, which are induced by a *global* Rydberg laser activating all atoms. If two qubits are within the *Rydberg radius* r_b, they become 'coupled,' enabling the application of a CZ gate by the Rydberg laser. Moving atoms between circuit stages couples different qubit pairs, resulting in a dynamic coupling graph. To avoid unintended 2-Q gates, other atoms must be *sufficiently separated* ($> 2.5r_b$). The global Rydberg laser requires less control and calibration, enhancing the *scalability* of FPQA compared to prior works [16, 28] where the laser *individually* address qubits for 2-Q gates.

We introduce Q-Pilot, the first scalable FPQA router drawing inspiration from FPGA placement and routing. Our approach, termed "routing with flying ancillas," involves qubit mapping akin to cell placement and the use of movable ancilla qubits to bridge fixed atoms, similar to FPGA routing. The advantages of flying ancillas include 1) high-parallelism circuit execution, 2) scalable compilation, and 3) no atom transfer required during computation. To boost parallelism and thus reduce circuit depth, we implement a high-parallelism generic router, dynamically arranging AODs and scheduling 2-Q gates. The router heuristically schedules as many parallel executable gates as possible in one laser stage up to AOD movement constraints. We also devise *application-specific* strategies: for each Pauli string in quantum simulation, we create multiple ancillas for a "root" qubit and employ graph algorithm to find the longest chain in the SLM array to perform the gates; for QAOA, we create ancilla per qubit instead of per gate, and leverage the commutation of gates to maximize parallel execution and reduce depth. Extensive experiments demonstrate that our FPQA compilation framework outperforms the best baseline, achieving 1.4×, 27.7×, and 6.7× smaller circuit depth for 100-qubit random, quantum simulation, and QAOA circuits.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06.
https://doi.org/10.1145/3649329.3658470

2 RELATED WORKS

Previously, research focused on fixed atom arrays with static coupling. Ref. [9] introduced the first compiler framework for this architecture, extending existing techniques for superconducting devices and addressing unique constraints, including long-range interaction restriction zones and sporadic atom loss. Ref. [20] further considers gate durations in compilation. Geyser (Ref. [27]) leverages native 3-Q operations by blocking 3-Q sub-circuits and re-synthesizing them.

Ref. [11] first considered atom movement, exploring a hypothetical architecture with '1D displacement' capability, more restrictive than FPQA. Ref. [32] presented the first optimal 2D FPQA compiler, formulating constraints and using an SMT solver for qubit mapping and routing. However, the solver-based method's scalability is limited by the exponential SMT solving. Their updated work [33] improves the scalability at some expense of optimality and reached the scale of about 100-qubit circuits within a day. Additionally, it is worth noting that they employ *atom transfer* operations, moving an AOD atom to an empty SLM trap when in proximity and vice versa. While atom transfer is already utilized in experiments [12], frequent transfers 'heat up' the atoms, potentially resulting in atom loss errors.

3 FLYING ANCILLAS

3.1 Motivating Example of Routing CZ

Revisiting the issue of the last gate in Fig. 1, (c) introduces an alternative using ancilla qubit q_A at p_3 instead of the SWAP: q_A is initialized to $|0\rangle$, the three-qubit initial state (in order $q_0 q_A q_2$) can be written as $a|000\rangle + b|001\rangle + c|100\rangle + d|101\rangle$. After the first CNOT, it becomes $a|000\rangle + b|001\rangle + c|110\rangle + d|111\rangle$. After the CZ, it becomes $a|000\rangle + b|001\rangle + c|110\rangle - d|111\rangle$. After the second CNOT, it is $a|000\rangle + b|001\rangle + c|100\rangle - d|101\rangle$, which is the same as the case where $\mathtt{CZ}(q_0, q_2)$ acts on the initial state. In this process, q_A acts as a 'fan-out' of q_0. However, note that it is only on the Z basis. Hence, this method's effectiveness hinges on the targeted 2-Q gate, specifically CZ in our case (and ZZ later on), but it is not universally applicable. Thus, we decompose other 2-Q gates using CZ or ZZ beforehand. Some previous works [15, 17] leveraged these fan-outs to reduce cost in circuit synthesis, but we apply them in routing because uniquely in FPQA, the fan-out qubits can move physically. If we rely on SWAPs for the routing, the depth increases by 3 because we need 3 CNOT for 1 SWAP, yet the new approach only increases depth by 2.

3.2 General Theory of Routing CZs with Ancillas

We prove a general result independent of the coupling graph. Given an arbitrary n-qubit state $\Psi = C_0|0\rangle + C_1|1\rangle + \ldots + C_{2^n-1}|2^n - 1\rangle$, and a set of qubit pairs C, applying $\mathtt{CZ}_{j,j'}\ \forall (j, j') \in C$ yields

$$\Psi' = \left(\prod_{(j,j')\in C} \mathtt{CZ}_{j,j'}\right)\Psi = \sum_{x=0}^{2^n-1} C_x \prod_{(j,j')\in C} (-1)^{x_j x_{j'}}|x\rangle, \quad (1)$$

where x_j is the j-th bit of x. We consider an alternative procedure as illustrated in Fig. 3 where we 1) apply transversal CNOTs from the n qubits to n fresh ancillas yielding Φ_1, 2) apply one of the four possibilities (2 choices of whether to $+n$ for 2 indices) $\mathtt{CZ}_{j(+n),j'(+n)}\ \forall (j, j') \in C$ yielding Φ_2, and 3) apply transversal CNOTs again yielding Φ_3. We prove that $\Phi_3 = \Psi' \otimes |0^n\rangle$, so our procedure is equivalent to applying the original CZs in Eq. 1.

For every basis state $|x\rangle$ appended with n fresh ancillas, applying transversal CNOTs flips the ancilla state to $|x\rangle$. Thus,

$$\Phi_1 = \left(\prod_{i=0}^{n-1} \mathtt{CNOT}_{i,i+n}\right)\left(\Psi \otimes |0^n\rangle\right) = \sum_{x=0}^{2^n-1} C_x|\overline{xx}\rangle, \quad (2)$$

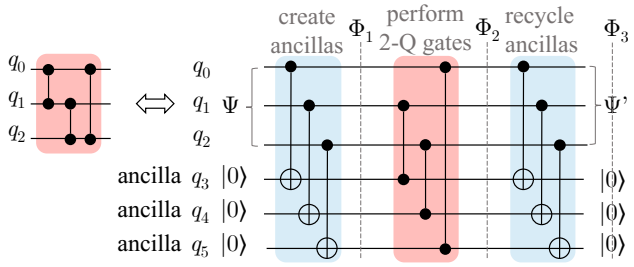

Figure 3: The general case of routing CZs with ancillas. The 3 CZs on the right can be executed simultaneously.

where an overhead line denotes the concatenation of bit-strings. Then, for every pair $(j, j') \in C$, we apply one of the 4 possible CZs,

$$\Phi_2 = \left(\prod_{(j,j')\in C} \mathtt{CZ}_{j(+n),j'(+n)}\right)\Phi_1 = \sum_{x=0}^{2^n-1} C_x \prod_{(j,j')\in C} \Big((-1)^{\overline{xx}_{j(+n)}\cdot\overline{xx}_{j'(+n)}}\ |\overline{xx}\rangle\Big) = \sum_{x=0}^{2^n-1} C_x \prod_{(j,j')\in C} (-1)^{x_j x_{j'}}\,|\overline{xx}\rangle, \quad (3)$$

where we use the fact that both the j-th bit (from the left) and the $(j+n)$-th bit of $\overline{xx}$ equals the j-th bit of x, similarly for j'. Applying transversal CNOTs, again, flips every state $|\overline{xx}\rangle$ back to $|x\rangle|0^n\rangle$, i.e.,

$$\Phi_3 = \left(\prod_{i=0}^{n-1} \mathtt{CNOT}_{i,i+n}\right)\Phi_2 = \Psi' \otimes |0^n\rangle, \quad (4)$$

which finishes our proof.

Note that CZ gates are commutable, so the ones in Eq. 3 can be applied in any order, which may unlock some freedom in scheduling. Moreover, for each $(j, j') \in C$, we have 4 possible CZs to use in Eq. 3 and many of them can be parallelized. For example, in Fig. 3, $n = 3$, and the original CZs are $C = \{(0, 1), (1, 2), (2, 0)\}$ which takes at least 3 steps. Using the procedure just presented, the CZs on $(0 + n, 1)$, $(1 + n, 2)$, $(2 + n, 0)$ can be scheduled to just one step.

3.3 Flying Ancillas in FPQA

The flying ancillas scheme proves particularly advantageous for FPQA over other QC platforms, owing to its high-fidelity movements. The most similar setting is in a multi-chain ion trap QPU [6], where chains of ions are laid out in 1D, and two chains can be moved to merge or split again. However, because there is no distinction between stationary and movable qubits like in FPQA, moving a regular qubit in the ion trap quantum computer has the same cost as moving an ancilla, so the flying ancilla scheme does not hold a big advantage. Additionally, the limited number of qubits available on ion trap QPUs discourages leveraging numerous ancillas. Flying qubits, typically optical, are also employed as communication resources between individual superconducting QPUs but face challenges, including a low interfacing fidelity of approximately 80% per flying qubit [21]. In contrast, in FPQA, the two extra CNOTs required by flying ancilla can achieve 99.5% fidelity, and the ancilla movement has negligible error [13]. Despite this high fidelity, the state-of-the-art FPQA compilation work [32] primarily utilizes only the movement of data qubits for routing, overlooking the potential advantages of routing via flying ancillas.

4 ROUTING FRAMEWORK

4.1 Overview

Given a target problem, the input values to the router are (1) the SLM array parameters (#rows, #columns, and locations), (2) the AOD

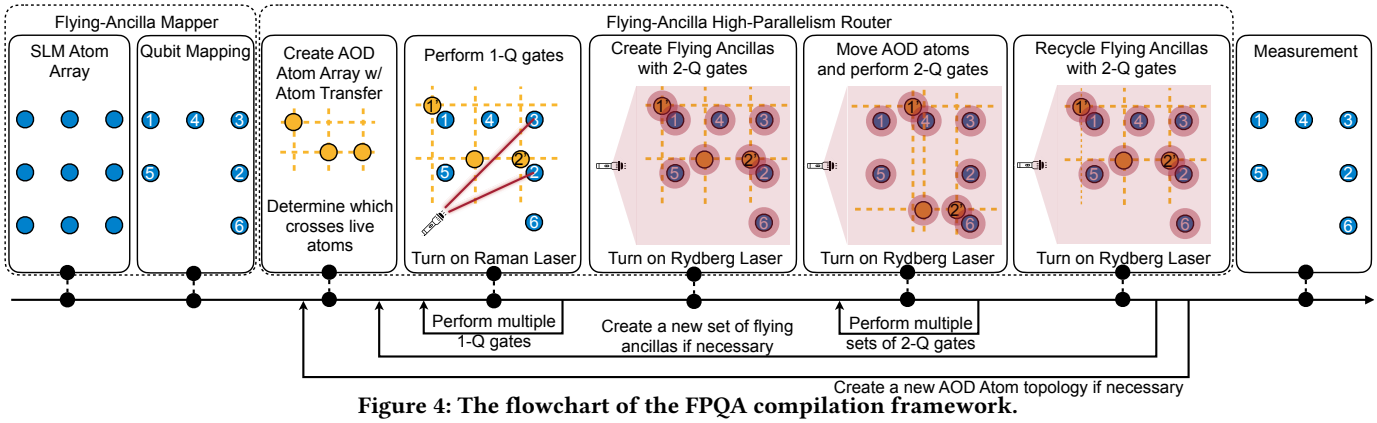

Figure 4: The flowchart of the FPQA compilation framework.

array parameters (#rows and #columns), and (3) the qubit mapping. We focus on routing in this work, so we simply map qubits in reading order throughout. We refer to them as *configurations* of the FPQA. Based on a configuration and the target problem, we leverage a high-parallelism router to generate an optimized schedule. A fast performance evaluator can efficiently return the corresponding performance metric or cost, including the number of 1-Q gates, 2-Q gates, the circuit depth, and movement distance, which are closely related to the circuit fidelity.

With this performance evaluator, our compiler also supports router-in-the-loop FPQA architecture design space exploration. We can use the evaluated cost as feedback to optimize a configuration that targets higher circuit fidelity iteratively. After certain epochs, the compiler will output the best configuration and optimized schedule.

4.2 Compilation of General Quantum Circuit

The compilation process is shown in Fig. 4. Given a quantum circuit, we first decompose the target circuit into 1-Q rotations and 2-Q `CZ` gates. Then, the gates are performed in alternating 1-Q and 2-Q stages. In the 1-Q stages, we turn on the individual addressable Raman laser to perform the desired gates on the target qubits. After all the available 1-Q gates are done, we move to 2-Q stages. In such stages, we select a set of `CZ` gates from the non-dependent front-layer of the circuit that can be performed in parallel. Then, we create flying ancillas from the control qubits and move the ancillas close to their corresponding target qubits. We turn on the Rydberg laser so that the ancillas will perform `CZ` gates with the target qubits. At last, we recycle these ancillas with `CNOT` gates and repeat this process. After all gates are done, we perform measurements and get results.

4.3 Customized Router for Quantum Simulation

For specific applications, we propose domain-specific routing strategies for higher parallelism. The first application is quantum simulation. To simulate the evolution under a Pauli string, the core part of the simulation algorithm works as follows: First, select a starting qubit inside the given Pauli string and then perform `CNOT`s on all pairs between the starting one and other qubits in the string.

We propose a longest path-based algorithm to compile this problem on the FPQA, described in Alg. 1. We configure the AOD array so that all ancilla qubits are on the *diagonal* of the grid and can be moved with the best flexibility. Then, we select the qubit i with the smallest index and fan out its state to all AOD ancilla qubits. To maximize the parallelism, we need to find the longest legal path in the dependency graph, where each qubit points to all other qubits in its lower-right corner, as shown in Fig. 5.

Algorithm 1: Customized router for quantum simulation

Data: List P: qubits in the Pauli string with non-I paulis
$s \leftarrow \emptyset$; Schedule for the compiled program
$P \leftarrow P \setminus P[0]$; $P[0]$ is the root qubit
$g \leftarrow \emptyset$; Directed compatibility graph. Two qubits are compatible if and only if there is a path between them.
for $q_i \in P$ **do**
  $g.nodes \leftarrow g.nodes \cup q_i$;
for $q_i \in P$ **do**
  for $q_j \in P \setminus q_i$ **do**
    if $q_j.row >= q_i.row$ & $q_j.col >= q_i.col$ **then**
      $g.edges \leftarrow g.edges \cup (q_i, q_j)$;
while $g \neq \emptyset$ **do**
  Find the longest path l in g;
  $s \leftarrow s \cup$ `GenerateSchedule`(l);
  $g \leftarrow g \setminus \{n | n \in l\}$;
Result: Schedule s

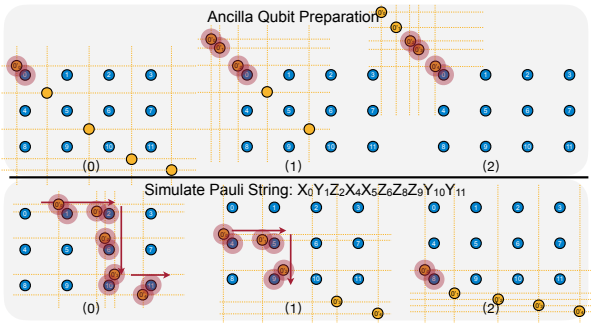

Figure 5: Q-Pilot routing quantum simulation circuits.

Given this longest path, we move the AOD ancilla qubits to their target SLM qubits and perform `CNOT`s in parallel. Further, those executed qubits will be removed from the candidate set, and the longest path-finding procedure will repeat until all gates are executed. Note that this longest path-finding can be implemented efficiently with dynamic programming. Compared to the generic router, which applies atom transfer to create and recycle ancilla qubits at each stage, this specialized router will maintain the states on the ancilla qubits across stages for one Pauli string, thus having a lower overhead.

4.4 Customized Router for QAOA

Another task that can be highly parallel is Quantum Approximate Optimization Algorithm (QAOA). In QAOA, we are given a graph, e.g., Fig. 6, and our target is to perform 2-Q gates on all its edges.

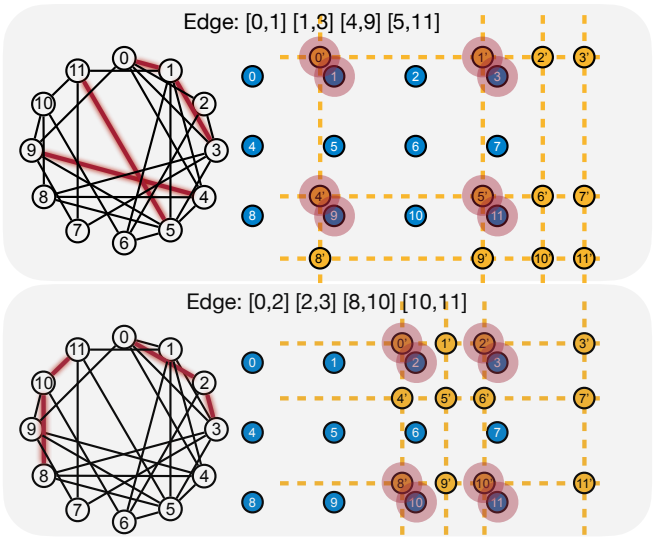

Figure 6: Scheduling of a QAOA circuit using Q-Pilot. The graph representation of the circuit is shown on the left. The edges correspond to interactions between two qubits.

First, we create one ancilla for each qubit. These ancillas will be recycled once the whole graph is done. Then, our router completes this task in a multi-stage way. Each stage will perform one or multiple 2-Q gates corresponding to some edges in the graph. We illustrate the detailed procedure of the first stage in Fig. 6. Among all the qubit pairs, we select the one with the smallest index as the highest-priority pair to begin with, e.g., (0',1). Since each AOD row and column must move simultaneously, we check which pair can be performed inside the same row, e.g., (1',3) is matched in this case. The rest of the AOD columns have already moved outside the SLM array. Then, they will not interact with any SLM atoms. Once the locations of all ancilla qubits in the first row are determined, other ancilla qubits on the rest of the rows can only move vertically due to the grid constraint. Then, we need to determine the vertical location of each row one by one. For the second AOD row, we found the best vertical location that can allow the most pairs to interact, e.g., in this case, we match two pairs (4',9) and (5',11). Note that any undesired interaction is illegal and thus should be avoided. This process will repeat until no rows can legally interact with any SLM atoms. Then, we can determine the locations of all AOD qubits and turn on the Rydberg laser to perform the parallel 2-Q gates. This greedy algorithm always tries to achieve the maximum matching on the first row and ultimately reaches a schedule with max parallelism.

So far, we have finished the first stage of the schedule with four 2-Q gates being performed. In the second stage, the highest-priority pair now becomes (0',2), and the same procedure as stage one can be applied to find a legal schedule with maximal parallelism. After t stages, the compilation flow ends with a t-stage legal schedule where all 2-Q gates are performed. Lastly, we recycle the ancillas and complete the task.

5 EVALUATION

5.1 Evaluation Methodology

Benchmarks. We utilize three benchmark types: random, quantum simulation, and QAOA circuits. Benchmarks were created for 5, 10, 20, 50, and 100 qubits. Random circuits were generated with Qiskit's `random_circuit` function, which randomly places 1-Q and 2-Q gates on qubits. The number of `CNOT` gates is set at 2x, 5x, 10x, 20x, and

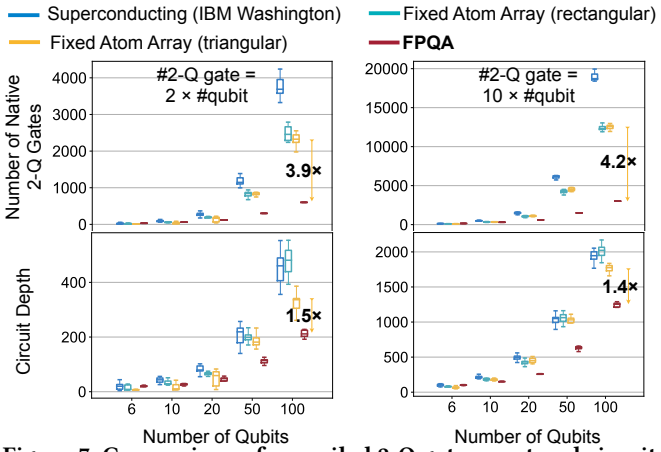

Figure 7: Comparison of compiled 2-Q gate count and circuit depth between Q-Pilot and the three baselines on random circuits. The random circuits vary in size, from 5-Q to 100-Q, and have 2-Q gate count between 2× and 10× qubit count.

50x the qubit count. Quantum simulation circuits were formed from 100 random Pauli strings. The probability p of a qubit having a Pauli operator X, Y, or Z varies from 0.1 to 0.5. QAOA circuits were constructed using ZZ gates between random qubit pairs. These pairs had an edge probability p of 0.1 to 0.5. We also designed specific QAOA circuits based on 3-regular and 4-regular graphs. These circuits also used 5, 10, 20, 50, and 100 qubits.

Baselines. 3 devices as baselines were chosen: the 127-qubit IBM Washington machine, a 16×16 square lattice, and a 16×16 triangular lattice of fixed neutral atoms, following Ref. [32]. The IBM machine features a heavy hexagon coupling graph. The square lattice's atoms connect to four nearest neighbors, while the triangular lattice's atoms connect to six. Qiskit's transpiler compiled benchmark circuits onto these devices at optimization level 3. Circuit depth, defined as the number of parallel 2-Q gate layers, was a key comparison metric, alongside the number of 2-Q gates in each *compiled* circuit for the baseline devices and Q-Pilot. Additionally, Q-Pilot was benchmarked against the solver-based compiler from Ref. [32], used for QAOA problems on 3- and 4-regular graphs. Comparisons included circuit depths and compilation times, with a 4,000s timeout (~an hour) set for the solver-based compiler due to its exponential runtime scaling.

5.2 Main Results

Results on random circuits. Fig. 7 shows the results of compiling random circuits. Compared with three baseline devices, for 100 qubits Q-Pilot shows an average of 4.2× reduction in the compiled 2-Q gate count, as well as an average of 1.4× reduction in compiled circuit depth compared with the best-performing baseline approach.

Results on quantum simulation circuits. Fig. 8 shows the results of compiling quantum simulation circuits. For Pauli probabilities 50%, Q-Pilot shows an average of 6.9× reduction in the compiled 2-Q gate count and an average of 27.7× reduction in compiled circuit depth compared with the best-performing baseline on 100-qubit circuits compared with the best baseline. Besides the random Pauli strings, we also test with the Pauli strings used in some molecule simulation problems [19]. As shown in Table 1, Q-Pilot shows an average 1.36× reduction in the 2-Q gate count and average 2.60× circuit depth reduction over the best baseline.

Results on QAOA circuits. Fig. 9 shows the results of compiling Max-Cut QAOA circuits for 4-regular graphs and random graph with

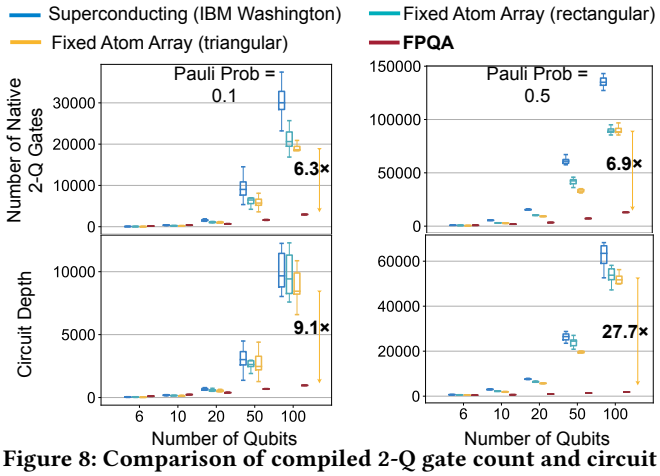

Figure 8: Comparison of compiled 2-Q gate count and circuit depth between Q-Pilot and the three baselines on quantum simulation circuits from 5-Q to 100-Q. The circuits are generated with Pauli probability $p = 0.1$ and 0.5.

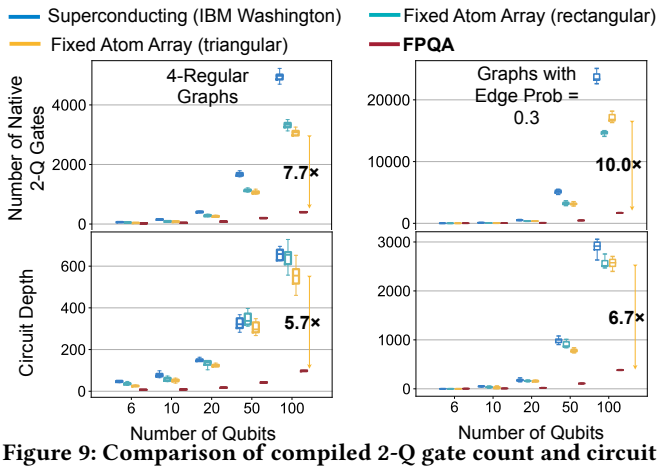

Figure 9: Comparison of compiled 2-Q gate count and circuit depth between Q-Pilot and the three baselines on QAOA circuits. The QAOA circuits vary in size, from 6-Q to 100-Q, and are generated with edge $p = 0.3$ and 4-regular graphs.

Table 1: Quantum Simulation for Molecule Pauli strings.

Benchmark	Device	H2	LiH_UCCSD	H2O	BeH2
Depth	FAA(rectangular)	76	2,772	31,087	43,919
	FAA(triangular)	61	2,052	26,189	37,314
	Superconducting	77	3,403	40,080	59,259
	Ours	61	849	7,585	10,617
#2Q Gate	FAA(rectangular)	82	3,577	41,306	58,720
	FAA(triangular)	73	2,616	35,353	51,699
	Superconducting	85	5,082	67,247	103,594
	Ours	94	2,130	20,966	29,518

Table 2: Comparison of Q-Pilot with solver based method.

Benchmark			6Q	10Q	20Q	50Q	100Q
3-reg.	runtime(s)	solver [32]	0.173	0.381	74.5	timeout	timeout
		iter-p [33]	0.509	2.16	14.6	966	timeout
		Ours	5.57E-3	9.89E-3	1.07E-2	7.52E-2	1.77E-1
	depth	solver [32]	3	3	3	-	-
		iter-p [33]	3	5	6	10	-
		Ours	5	7	11	24	45
4-reg.	runtime(s)	solver [32]	18.1	3.93E3	timeout	timeout	timeout
		iter-p [33]	0.852	2.64	23.4	2.34E3	timeout
		Ours	6.25E-3	9.31E-3	2.10E-2	7.23E-2	3.42E-1
	depth	solver [32]	5	5	-	-	-
		iter-p [33]	5	6	8	15	-
		Ours	6	9	15	32	60

edge occupancy 30%. Q-Pilot again shows an average of 10.0× reduction in compiled 2-Q gate count and an average of 6.7× reduction in the compiled circuit depth.

Comparison with the Solver-Based Compiler. As illustrated in Table 2, we compare Q-Pilot against the solver-based methods [32, 33] in compiling QAOA circuits for regular graphs. Ref. [33] relaxes the formulation of Ref. [32] to tradeoff compilation time and quality. While the these method achieve better solutions, they struggles with larger problems, often failing to find a solution within an hour due to its exponential runtime scaling. In contrast, Q-Pilot efficiently compiles all these problems in under 1 second, with the compiled circuit depth not exceeding 4× the optimal depth.

5.3 Analysis

Impact of Array Size on Circuit Depth. Fig. 10 shows how array sizes affect the compiled circuit depth. We organized the qubits into rectangular arrays of varying widths (8, 16, 32, 64, 128), with the optimal array widths marked by stars in the figure. Optimal array widths vary across different problems, highlighting a trade-off between greater parallelism within a row and across different rows. Specifically, in Fig. 10, we observe that QAOA circuits achieve optimal performance with large array width (128), while random circuits and quantum simulation problems are best served with moderate array widths (64 or 32 in our study). The insight here is while larger array widths offer more parallel execution paths, they might not always correspond to increased efficiency for all types of problems, possibly due to overheads or specific characteristics of the circuit structure.

How does the 2-Q gate error rate affect the overall error rate? Fig. 11 (a) shows the relation between the overall error rate and the 2-Q gate error rate. We model the circuit error with the equation introduced [32], where ϵ is the overall error rate:

$$\epsilon = 1 - f_2^{NT} f_1^{G_1} \exp\left(-N \frac{\sum_i T_0 \sqrt{D_i}}{T_2}\right), \tag{5}$$

N is the maximum number of qubits used (including AOD and SLM), and T is the circuit depth. G_1 is the number of 1-Q gates. f_1 and f_2 are the fidelity of 1-Q and 2-Q gates, respectively. T_2 is the coherence time of the qubit, and T_0 is the characteristic time of atom movement. D_i is the maximum distance atoms moved in stage i. In our estimation, we choose $f_1 = 99.9\%$, $T_2 = 1.5$s, and $T_0 = 300\mu$s [32]. The three benchmarks used here are 1) quantum simulation circuits with 5 qubits and 100 non-trivial Pauli strings with $p = 0.1$, 2) random 5Q circuits with an average of two 2-Q gates per qubit, and 3) QAOA circuits for random 3-regular graphs. The error rates are below 0.5 when the 2-Q gate has an error rate below 10^{-3}.

What is the distribution of the parallelism? Fig. 11 (b) shows the percentage of stages with the number of 2-Q gates simultaneously executed for QAOA problems. The average parallelism of 20Q, 50Q, and 100Q problems are 3.32, 4.13, and 4.90, respectively. As the problem scales up, the parallelism of the problem is also increased.

Whether the application-specific compilers bring better performance for quantum simulation and QAOA? Fig. 12 shows the advantage of the domain-specific compiler compared to the general compiler. For quantum simulation, the domain-specific compiler reduces the 2-Q gate count by 1.5× and the circuit depth by 8.8×. For QAOA, the domain-specific compiler reduces the 2-Q gate count by 2.8× and the circuit depth by 10.1×. The advantages come from domain-specific heuristics that minimize the circuit depth.

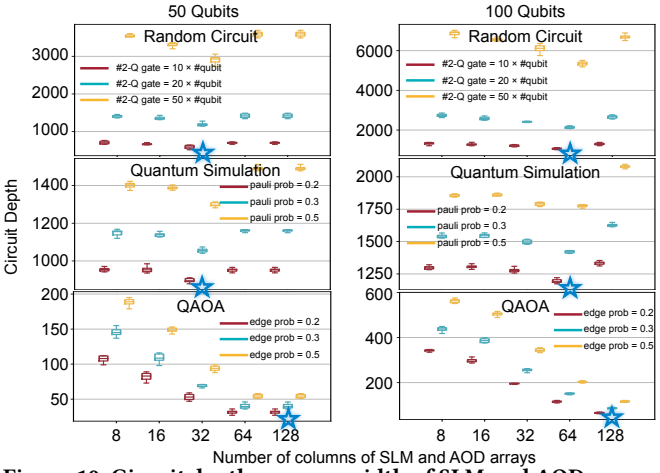

Figure 10: Circuit depth vs array width of SLM and AOD arrays in FPQA. The three benchmark circuits are shown here for 50 qubits and 100 qubits. The star in each graph marks the optimal array width for the smallest circuit depth.

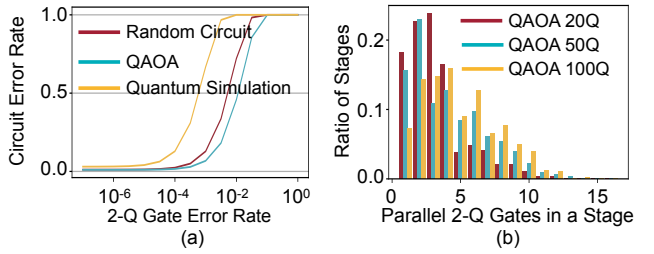

Figure 11: (a) Overall error rate vs. 2-Q gate error rate for random 6Q circuits with two 2Q gates per qubit, QAOA circuits based on random 3-regular graphs, and 5Q quantum simulation circuits with 100 Pauli strings and $p = 0.1$. (b) Ratio of total stages vs number of parallel 2-Q gates in a stage using Q-Pilot on QAOA circuits with 20-Q, 50-Q, and 100-Q.

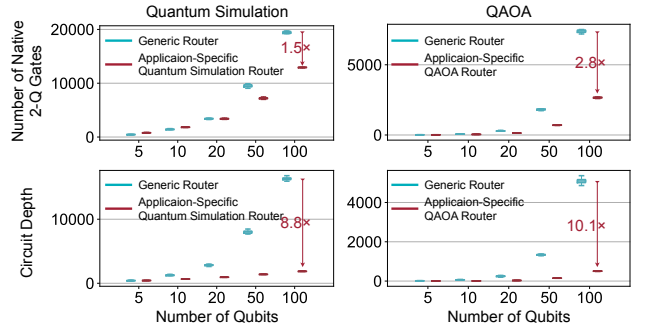

Figure 12: Advantage of our application specific Quantum Simulation and QAOA router comparing to the generic router.

How scalable is the Q-Pilot? We test Q-Pilot with a large number of qubits to show its scalability. For the QAOA problem, we choose random graphs with edge $p = 0.5$. It takes 1.51s, 10.75s, and 129.50s to compile 500, 1000, and 2000 qubits. For quantum simulation problems, we choose 100 random Pauli strings. It takes 6.91s, 14.28s, and 30.48s to compile 500, 1000, and 2000 qubits. We generate random circuits with a depth of 10 for general circuits, and it takes 2.64s, 8.70s, and 32.31s to compile 500, 1000, and 2000 qubits. The fast speed proves that Q-Pilot is scalable and can handle large-scale problems.

6 CONCLUSION AND OUTLOOK

We design a compilation framework for FPQA with movable atoms, enabling flexible qubit mapping and efficient 2-Q gate execution. Our approach includes a versatile router for enhanced parallelism in quantum simulations and QAOA. Future directions involve refining search heuristics for better solutions, exploring quantum error correction in FPQA-based QPUs, and using compiler insights for hardware advancements like multi-functional FPQA zones.

ACKNOWLEDGEMENT

This work is partially supported by NSF grant 442511-CJ-22291, MIT-IBM Watson AI Lab, and Qualcomm Innovation Fellowship. The authors would like to thank Dolev Bluvstein, Mikhail D. Lukin, and Hengyun Zhou for valuable discussions on neutral atom arrays.

REFERENCES

[1] https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two.
[2] https://www.rigetti.com/.
[3] https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html.
[4] https://spectrum.ieee.org/tech-talk/computing/hardware/intels-49qubit-chip-aims-for-quantum-supremacy.
[5] https://www.quera.com/aquila.
[6] https://ionq.com/posts/august-25-2021-deep-dive-reconfigurable-multicore-quantum-architecture.
[7] M. Alam et al. An efficient circuit compilation flow for quantum approximate optimization algorithm. DAC'20.
[8] M. Alam et al. Noise resilient compilation policies for quantum approximate optimization algorithm. ICCAD'20.
[9] J. M. Baker et al. Exploiting long-distance interactions and tolerating atom loss in neutral atom quantum architectures. ISCA'21.
[10] D. Bluvstein et al. A quantum processor based on coherent transport of entangled atom arrays. *Nature* 604, 7906 (2022), 451–456.
[11] S. Brandhofer et al. Optimal mapping for near-term quantum architectures based on Rydberg atoms. ICCAD'21.
[12] S. Ebadi et al. Quantum optimization of maximum independent set using Rydberg atom arrays. *Science* 376, 6598 (2022), 1209–1215.
[13] S. J. Evered et al. High-fidelity parallel entangling gates on a neutral atom quantum computer. *Nature* 622 (2023), 268–272.
[14] H. Fan et al. Optimizing quantum circuit placement via machine learning. DAC'22.
[15] P. Gokhale et al. Quantum fan-out: circuit optimizations and technology. QCE'21.
[16] T. M. Graham et al. Multi-qubit entanglement and algorithms on a neutral-atom quantum computer. *Nature* 604, 7906 (2022), 457–462.
[17] P. Høyer et al. Quantum fan-out is powerful. *Theory of Computing* 1, 5 (2005).
[18] G. Li et al. Tackling the qubit mapping problem for NISQ-era quantum. ASPLOS'19.
[19] G. Li et al. Paulihedral: a generalized block-wise compiler optimization framework for quantum simulation kernels. ASPLOS'22.
[20] Y. Li et al. Timing-aware qubit mapping and gate scheduling adapted to neutral atom quantum computing. *TCAD* 42, 11 (2023).
[21] P. Magnard et al. Microwave quantum link between superconducting circuits housed in spatially separated cryogenic systems. *PRL* 125 (2020), 260502.
[22] D. Maslov et al. Quantum Circuit Placement. *TCAD* 27, 4 (2008), 752–763.
[23] Abtin Molavi et al. Qubit mapping and routing via MaxSAT. MICRO'22.
[24] P. Murali et al. Full-stack, real-system quantum computer studies: architectural comparisons and design insights. ISCA'19.
[25] M. A. Norcia et al. Iterative assembly of ^{171}Yb atom arrays in cavity-enhanced optical lattices. arXiv:2401.16177.
[26] S. Park et al. A fast and scalable qubit-mapping method for noisy intermediate-scale quantum computers. DAC'22.
[27] T. Patel et al. Geyser: a compilation framework for quantum computing with neutral atoms. ISCA '22.
[28] M Saffman. Quantum computing with atomic qubits and Rydberg interactions: progress and challenges. *Journal of Physics B* 49, 20 (oct 2016), 202001.
[29] M. Y. Siraichi et al. Qubit allocation. CGO'18.
[30] B. Tan et al. Optimal layout synthesis for quantum computing. ICCAD'20.
[31] B. Tan et al. Optimal qubit mapping with simultaneous gate absorption. ICCAD'21.
[32] B. Tan et al. Qubit mapping for reconfigurable atom arrays. (2022). ICCAD'22.
[33] D. B. Tan et al. Compiling quantum circuits for dynamically field-programmable neutral atoms array processors. *Quantum* 8 (2024).
[34] R. Wille et al. Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. DAC'19.
[35] T.-A. Wu et al. A robust quantum layout synthesis algorithm with a qubit mapping checker. ICCAD'22.
[36] X. Zhou et al. A Monte Carlo tree search framework for quantum circuit transformation. ICCAD'20.
[37] A. Zulehner et al. Efficient mapping of quantum circuits to the IBM QX architectures. DATE'18.