# Design of an FPGA-Based Neutral Atom Rearrangement Accelerator for Quantum Computing

Xiaorang Guo, Jonas Winklmann, Dirk Stober, Amr Elsharkawy, and Martin Schulz
Chair of Computer Architecture and Parallel Systems, Technical University of Munich, Garching, Germany
Email: {xiaorang.guo, jonas.winklmann, dirk.stober}@tum.de, {elsharka, schulzm}@in.tum.de

***Abstract*—Neutral atoms have emerged as a promising technology for implementing quantum computers due to their scalability and long coherence times. However, the execution frequency of neutral atom quantum computers is constrained by image processing procedures, particularly the assembly of defect-free atom arrays, which is a crucial step in preparing qubits (atoms) for execution. To optimize this assembly process, we propose a novel quadrant-based rearrangement algorithm that employs a divide-and-conquer strategy and also enables the simultaneous movement of multiple atoms, even across different columns and rows. We implement the algorithm on Field Programmable Gate Arrays (FPGAs) to handle each quadrant independently (hardware-level optimization) while maximizing parallelization. To the best of our knowledge, this is the first hardware acceleration work for atom rearrangement, and it significantly reduces the processing time. This achievement also contributes to the ongoing efforts of tightly integrating quantum accelerators into High-Performance Computing (HPC) systems.**

**Tested on a Zynq RFSoC FPGA at 250 MHz, our hardware implementation is able to complete the rearrangement process of a 30×30 compact target array, derived from a 50×50 initial loaded array, in approximately 1.0 $\mu s$. Compared to a comparable CPU implementation and to state-of-the-art FPGA work, we achieved about 54× and 300× speedups in the rearrangement analysis time, respectively. Additionally, the FPGA-based acceleration demonstrates good scalability, allowing for seamless adaptation to varying sizes of the atom array, which makes this algorithm a promising solution for large-scale quantum systems.**

***Index Terms*—Quantum Computing, Neutral Atoms, FPGA, Atom rearrangement**

## I. Introduction

Quantum computing, as a revolutionary computing approach, promises computational complexity advantages for particular problems [1], [2]. Among all the different technologies, also referred to as physical modalities, that can be used to realize quantum computing, neutral atoms are particularly interesting since they can provide better scaling characteristics and longer coherence times [3].

In neutral atom quantum computers, preparing for quantum operations involves stochastically loading atoms into a 2D array of optical traps, with a filling probability of approximately 50% [4]. Then, information is commonly extracted through fluorescence imaging [5], and an atom detection algorithm analyzes these images to ascertain the occupancy information of the traps. However, for reliable operation, all atoms must be arranged in a defect-free, predetermined geometric array. To achieve this fully-occupied atomic array, an atom rearrangement algorithm, or atom sorting, is essential [6]. Starting from a random initial condition, such an algorithm develops a rearrangement schedule of the atoms to target positions, ensuring the elimination of defects in the array structure. The scheduled movements are then sent to an Arbitrary Waveform Generator (AWG) to control the Acousto-Optic Deflector (AOD) to manipulate the qubits. Fig. 1 summarizes the workflow of the neutral atom quantum computer.

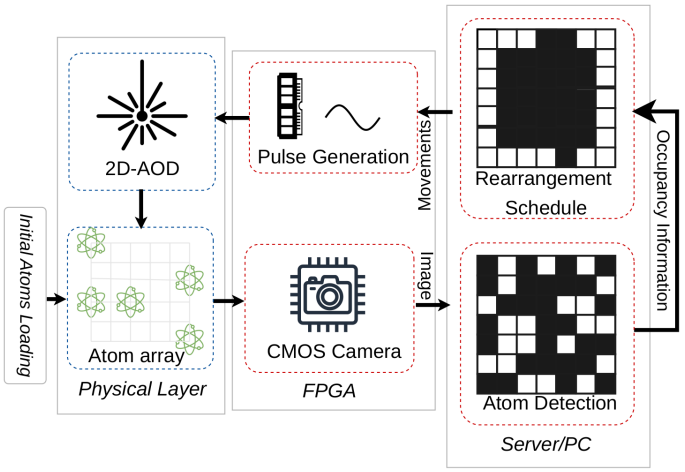



Fig. 1. A typical workflow of neutral atom quantum computers. The image of the atom array is transformed into a binary representation, where black dots denote occupied areas, while white dots stand for positions where no atom is detected. This binary format serves as the input for the rearrangement algorithm. After making a schedule, the details of movements will be sent to the AWG, whose pulses control the AOD to tune the atoms.

Although neutral atoms benefit from their long coherence times [7], the runtime for atom rearrangement in scaled-up systems with mid-circuit measurements remains a challenge [6]. To date, significant efforts have been directed towards the development of faster rearrangement algorithms utilizing multi-mobile tweezers to maximize simultaneous atom moves, achieving a higher level of parallelization [6], [8], [9]. However, one of the major challenges to enabling scalable and efficient neutral atom control systems is mitigating the communication overhead among system components. Fig. 2(a) demonstrates a control system architecture where camera connection via CoaXpress protocol [10] and signal generation using AWG are performed on Field Programmable Gate Arrays (FPGAs), but

This work was funded by the German Federal Ministry of Education and Research (BMBF) under the funding program *Quantum Technologies - From Basic Research to Market* under contract number 13N16087, as well as from the Munich Quantum Valley (MQV), which is supported by the Bavarian State Government with funds from the Hightech Agenda Bayern.

978-3-9826741-0-0/DATE25/© 2025 EDAA

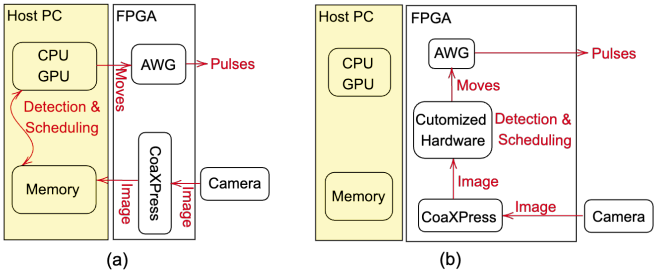



Fig. 2. Architectures of atom control systems. (a) Current typical system structure: Detection and rearrangement processes are performed using CPUs or GPUs, where communications between components are needed. (b) Optimal (long-term) structure: All functional blocks are seamlessly integrated into FPGAs, with detection and rearrangement implemented via customized hardware.

atom detection and rearrangement analysis are processed on a CPU/GPU located in a server. This cumbersome transmission process leads to significant overheads. To overcome this limitation, we propose a fully *FPGA-based control system* as shown in Fig. 2(b), which includes our focused rearrangement module.

In addition to reduced overhead, FPGA-based rearrangement offers significant acceleration due to the inherent parallelism of the process, where the atom array's four quadrants follow a uniform rearrangement schedule. Building on this, we propose a Quadrant-based Rearrangement Method (QRM) to further accelerate the process on the FPGA's Programmable Logic (PL) side. This approach not only speeds up the rearrangement algorithm, but also reduces communication overhead with other system components.

This paper mainly focuses on the QRM algorithm and its FPGA implementation, and has the following contributions:

- We identify a uniform moving pattern within an atom array during the rearrangement process and introduce a quadrant-based acceleration-friendly algorithm.
- We develop, to the best of our knowledge, the first work to implement the rearrangement algorithm on the PL part of an FPGA, achieving high parallelization and very low processing latency.
- We show that our FPGA accelerator achieves about 54× speedup in terms of processing latency compared to a CPU implementation when tested using a 50×50 atom array. Furthermore, compared to the recent FPGA rearrangement design [6] (only implemented on the ARM core on the FPGA), we achieve a speedup of around 300×.

## II. Background

### A. Neural Atom Basics

Neutral atoms trapped in optical lattices present a promising platform for implementing quantum computers. In this approach, qubits are encoded in individually trapped atoms, with interactions mediated via their electronically highly excited Rydberg states [5], [11]. By applying crossed laser beams to the qubits, we induce periodic transitions between the two quantum states, '0' and '1', resulting in gradual changes in the probability of the atom's state. Consequently, the atom can enter any superposition state at specific time points. Entanglement between two qubits can be achieved through the Rydberg blockade mechanism [7]. By simultaneously manipulating two atoms and exciting their electrons to a highly energetic state, the influence between them prevents exactly one of them from entering the excited state, thus inducing entanglement [12].

For quantum computing purposes, we need to generate a regularly spaced atom array that can be prepared by a 2D Spatial Light Modulator (SLM) [13]. SLMs are able to guide a cloud of cold atoms into targeted traps to load atoms. However, this process is probabilistic, with a loading probability (each optical trap in the array can successfully capture and hold a single atom during the loading process) of around 50%. Therefore, we need a rearrangement process to construct a defect-free array for execution.

### B. Hardware Constraints

To control the movements of atoms, we provide our 2D-AOD setup with several RF frequencies per dimension, thereby allowing us to generate a grid of movable tweezers. This kind of multi-tweezers system is able to move atoms at the same time when they are to be moved towards the same direction with the same step size. While this enables us to parallelize the atom rearrangement, it greatly complicates the process. Since the incident beam passes through two subsequent AODs, we can only choose a set of rows and columns instead of individual traps. A trap will be generated at every coordinate for which both the row and column are selected. If we want to generate a trap at location $(x_1, y_1)$ and another at $(x_2, y_2)$ by selecting $x_1$, $x_2$, $y_1$, and $y_2$, then there will also be traps at $(x_1, y_2)$ and $(x_2, y_1)$. In situations where this constitutes a problem, the two atom sites will have to be addressed in separate moves. Having generated such a set of traps, we can shift the contained atoms in lockstep to a target location.

## III. Methodology

To reduce the processing time of the atom rearrangement from the algorithm perspective, we first analyze the typical rearrangement procedure and identify the inherent parallel moving pattern. Then, we propose an efficient and acceleration-friendly algorithm to reduce the time frame for schedule analysis.

### A. Typical Rearrangement Procedure

Atoms are loaded stochastically into a large array, and the target area is typically located in the center (due to physical fidelity constraints). The key point of the rearrangement procedure is to reposition (compress) atoms to the array's center. Fig. 3 illustrates the compression mechanism of a typical algorithm using an $8 \times 8$ square lattice as a demonstration. Initially, we construct a matrix to represent the lattice, where each cell can either be occupied by an atom (indicated by a black dot) or remain empty (indicated by a white dot). When loading the atoms, we apply a filling factor of 50%, aiming for a defect-free array size of $4 \times 4$.

The rearrangement process involves two types of movements: horizontal moves (in Fig. 3, Move 1 to Move 4) and vertical moves (Move 5 to Move 6). In the horizontal movement

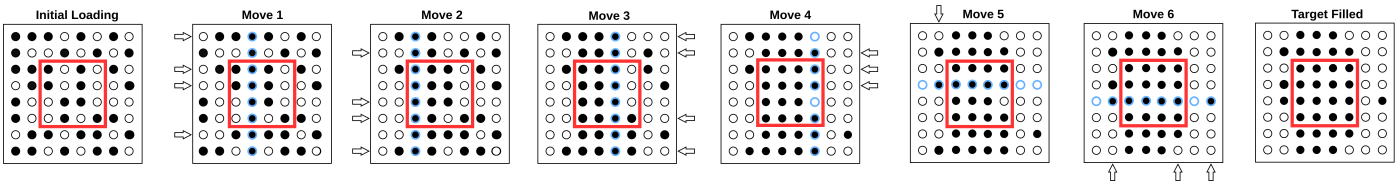



Fig. 3. Illustration of the typical rearrangement algorithm. The blue emphasized line indicates the target line for this move step. Arrows represent the atoms being moved in this step, with the direction of arrows indicating the direction of movement. The red square stands for the target filling area. Within one "Move" block, we can have multiple simultaneous moves. For instance, in "Move 1", there are empty holes in Rows 1, 3, 4, and 7, so we move all atoms positioned to the left of each hole, shifting them one step to the right.

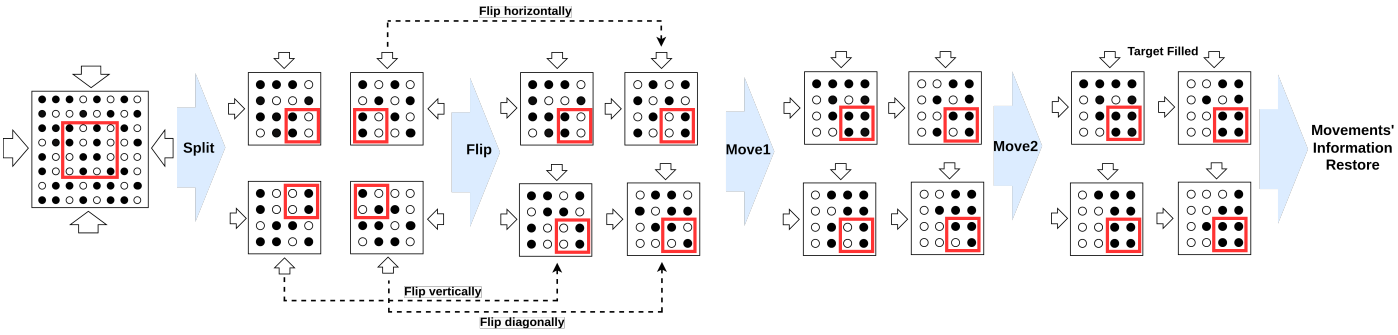



Fig. 4. QRM rearrangement schedule. By splitting the atom array and performing specific flip operations, we can apply a unified rearrangement method to each quadrant. This method provides an inherent acceleration on FPGA.

procedure, we begin by identifying the center column of the array. Next, we locate the empty sites within this central column. Subsequently, we shift the atoms from the left or right (depending on whether the column we select is in the right or left part) towards these empty traps, effectively filling the gaps. Notably, with the multi-tweezer hardware employed in our system, we are able to move multiple atoms simultaneously, only if they are all displaced in the same direction with the same step size. This protocol is also illustrated in Fig. 3. In "Move 1", a total of nine atoms, arranged in four distinct lines (indicated by the white arrows on the left), are shifted toward the blue lines. Once the atoms have been adjusted within the center column, we extend the same process to the adjacent columns, ensuring a consistent filling procedure. This sequence of actions continues until we reach the boundary of the targeted filling area. For the vertical moves, we utilize the same schedule strategy, but move rows in this case. This process results in the seamless filling of the holes and a parallel movement, generating defect-free columns throughout the process.

### *B. Quadrant-based Rearrangement Method (QRM)*

By analyzing the typical moving procedure, we have observed a consistent atom movement pattern across the four quadrants in this process, suggesting a uniform mechanism. Compressing all the atoms to the center of the large array is essentially compressing the atoms into the corner of each quadrant. Therefore, when we flip array quadrants via a specified direction, we can reuse the same moving schedule on each quadrant, which introduces an intrinsic parallelization with a factor of four. Thus, we propose a quadrant-based rearrangement method, as shown in Fig 4. In QRM, we first split the large atom array into four parts and flip each part in different directions to put the target area at the bottom left corner. Then, a unified moving schedule is applied to the four parts. After the target area is filled, we restore the moving information to its original position.

QRM suggests a promising parallelization architecture for an FPGA implementation: we can create four pathways, each aligned to a quadrant. By independently processing the schedule analysis within each quadrant with a single functional call, we can realize an inherent acceleration.

## IV. FPGA Implementation

Having discussed the moving schedule, we detail the implementation of the QRM rearrangement algorithm on a Zynq Radio Frequency System-On-Chip (RFSoC) FPGA board [14] for acceleration in this section.

### *A. System Overview of QRM Accelerator*

The system architecture used for implementing our QRM algorithm is an FPGA board consisting of three main components: the Processing System (PS) component, the PL component, and DDR memory. The PS part hosts an ARM core equipped with a Linux system responsible for handling memory access and interconnect logic, which we use to control I/O and to orchestrate the rearrangement, while the PL part of the board is used to hold our customized rearrangement accelerator (described in Sections IV-B and IV-C). The DDR memory is used for communication between PS and PL and holds both data used and produced by the rearrangement procedure. To enhance data transmission efficiency, we pack 1024-bit data into one transmission packet to move the data from DDR memory into our accelerator with minimal transmission overhead.

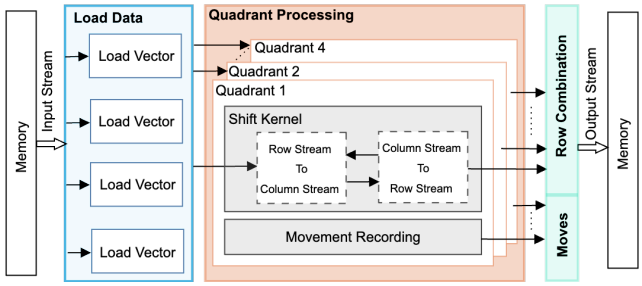



Fig. 5. Complete dataflow of the HLS-accelerated rearrangement module.

In this work, the whole processing procedure is controlled via a Python application programming interface (API), which we implement to run on the ARM part of our target system. After the user loads the initial data into the memory, they can trigger the rearrangement step. For this, we then configure an AXI-based transmission to transfer the binary data, containing the output of the atom detection unit in the form of a bitfield, via the internal bus into our rearrangement module. Inside the module, we have a buffer to temporarily store the data. Once the transmission is finished, we kick off the rearrangement algorithm on the PL of the FPGA, which then stores the needed atom movements back into the memory. When the rearrangement process is finished, the final matrix and movement details are transferred back and analyzed on the PS side.

### B. Rearrangement Module

The proposed rearrangement module for the QRM algorithm, illustrated in Fig. 5, features inherent parallelization within the four quadrants to optimize performance. Designed in a dataflow fashion, the accelerator is fully pipelined and comprises three main modules: the Load Data Module (LDM), the Quadrant Processing Module (QPM), and the Output Concatenation Module (OCM).

Initially, the input data stream is processed by the LDM, where four *Load Vector* units divide the large atom array into smaller arrays, each representing a quadrant. During this stage, the flip operation is automatically performed to prepare the data. These smaller arrays are then passed to the QPM, enabling independent and parallel processing of each quadrant. Within each quadrant, the core operation, referred to as the *Shift Kernel*, determines the schedule for atom movements. This process will be explained in detail in Section IV-C. Accompanying the *Shift Kernel* is a movement recording unit responsible for tracking atom movements and restoring their original positions from the quadrant-based locations. This unit records essential details, such as the original location of atoms, their directional shifts, and the number of steps taken. Once the processing steps are completed, the movement records and the final defect-free array are consolidated into a single output stream within the OCM. This integrated output stream is then transmitted to memory for further use.

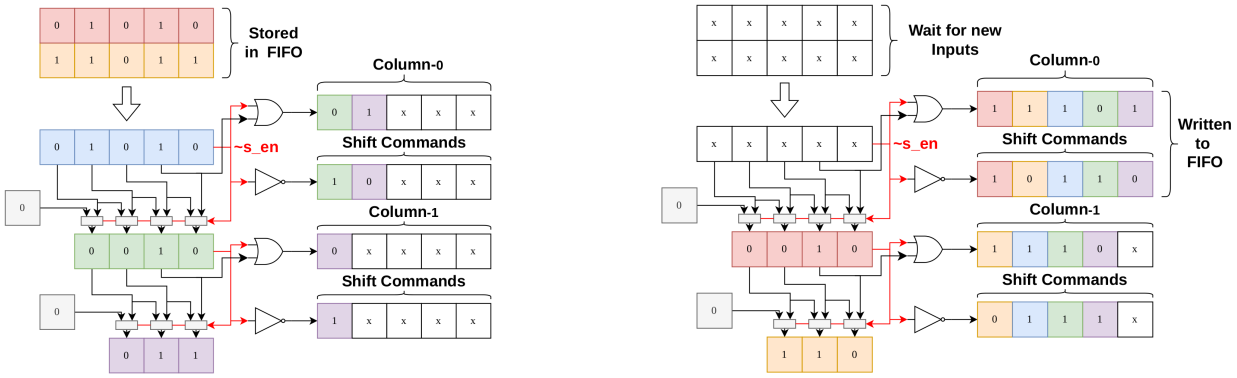
### C. Shift Kernel

When migrating the rearrangement process onto FPGA, one of the main challenges is *how to efficiently perform the movements and make the movement schedule*? To solve this problem, we employ shift registers on FPGA to represent the atom movements. By interpreting columns as rows, we design a fully pipelined shift kernel that can process both vertical and horizontal movements with minimum latency. Since different quadrants have already followed the specific flip operation in the LDM, we can reuse the same shift architecture for all quadrants.

In this unit, every row/column is represented as a bit vector. Each quadrant has $Q_w = \frac{W}{2}$ rows, where W represents the length (size) of the initial array. The architecture can inspect every element of every row in a fully pipelined way; thus, we are able to start processing a new row every clock cycle. This results in a total latency of $Q_w$ when starting to process all rows. Inside a single quadrant, we first shift row-wise and then column-wise and repeat both steps until the center is filled. In this case, for one iteration round, the total latency of the shift kernel is $2 \times Q_w$ plus the processing time of a single row.

Fig. 6 explains the working principle of the shift kernel in detail with a row-wise example of the northwest (NW) quadrant. In this example, we use an atom array with a size of $10 \times 10$, so the sub-array for each quadrant is $5 \times 5$. Each row (represented by a specific color) is first fetched from the input queue for analysis, and then every element of each row is inspected step-by-step. As also shown in the figure, different rows can be analyzed simultaneously by benefiting from the pipeline mechanism to maximize the throughput and parallelization of our design. We use the *shift commands* buffer to record which atoms are shifted and the *column buffer* to record the states of the atoms after the shifts. After fetching one row, we check its lowest significant bit (LSB). If it is set, no shift command has to be issued, and we store a '1' (the value of the LSB) into the respective *column buffer* and a '0' (no shift) into the *shift commands* buffer. After processing the LSB, we shift the entire row by one to the right to check the next bit in the next stage, and we can read a new row for the current stage. Furthermore, to be able to prevent unnecessary shifts far from the center, we introduce a manual-control mechanism. In this mechanism, we can manually set the $s_{en}$ signal (shown in Fig. 6) to '0' to block the row elements from shifting.

After processing each quadrant, the four *shift command buffers* are written into a large FIFO and processed by the *Row Combination Unit*. All four command buffers are processed at the same time, and it is also statically known which shift commands finish at which time. The logic of merging the shift commands follows the rules explained earlier. First, all '1's inside the shift command vector of a single quadrant can be executed simultaneously. For example, in Fig. 6(b), the red, blue, and green elements in Column 0, which all have a '1' in the *shift commands*, can be shifted at the same time. Secondly, we analyze the shifts of the NW and southwest (SW) quadrants at the same time and perform the initial shifts in the same command since they contain the same shifts for the most central column from the west. Similarly, the shifts of the quadrants northeast (NE) and southeast (SE) can be merged as well to shift once from the east. The column-wise *shift commands*




(a) Shift Unit after 3 cycles

(b) Shift Unit after $Q_w + 1$ cycles

Fig. 6. Row-wise to column-wise shift process for a $10 \times 10$ input atom array ($Q_w = 5$). (a) State of the shift unit after 3 cycles. The shift kernel is processing the first three rows, and each bit in the row buffer is transferred to the corresponding *column buffer*. The *shift commands* buffer records whether the shifted value is '0'. (b) Status after $Q_w + 1$ cycles. At this time point, the purple, green, and blue rows from (a) have finished processing. *Column-0* ('11101') represents the original right-most bit in each row, demonstrating the row-to-column transition process.

represent shifts from the south and north. Thus, the quadrants NE and NW are merged, as are SE and SW. During this process, empty shifts are removed from the final schedule to reduce the total number of commands.

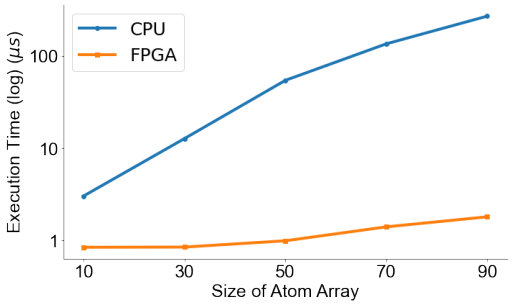
## V. Evaluation

### A. Experimental Setup

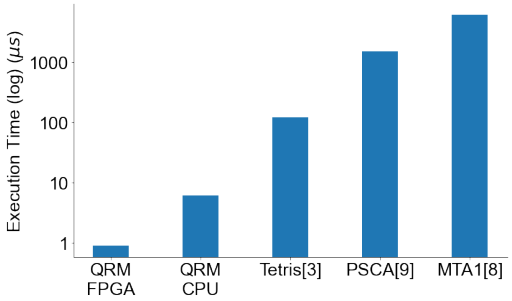
In this work, our rearrangement accelerator is programmed on the FPGA using an HLS-compatible C++. The design is synthesized and generated with Xilinx Vitis HLS 2022.2, and we use Xilinx Vivado 2022.2 to implement and deploy the entire project on the RFSoC FPGA at a clock frequency of 250 MHz. Additionally, we conduct a comparative analysis by running the C++ algorithm on the CPU. The CPU is an 11th-generation Intel Core i7 operating at 2.8GHz. As input data, we use a randomly generated matrix representing a random distribution of atoms (instead of using actual detection data from image analysis), as the randomized images have the same distribution on average and, hence, are sufficient for the analysis of our hardware design.

### B. Execution Time

Fig. 7(a) illustrates a performance comparison between the CPU-based QRM and its FPGA-accelerated version when varying the initial atom array sizes from $10\times10$ to $90\times90$ with a step size of 20. As demonstrated in Fig. 7(a), employing our FPGA-based solution yields a substantial speedup, achieving approximately $54\times$ faster processing when the array size is 50, and up to $134\times$ acceleration for an array size of 90. Moreover, a noteworthy aspect of our design is its scalability, which is evident in the comparison of performance trends with the CPU version. Our design exhibits a more moderate increase in run time as the size of the initial array grows (0.8 $\mu s$ with a size of 10, 1.0 $\mu s$ with a size of 50, and 1.9 $\mu s$ with size of 90), indicating its suitability for large-scale quantum computation scenarios. It's also important to note that the latency of our design is not directly dependent on the target area we aim to fill. Instead, it correlates solely with the initial size of the array and the number of iterations required by the algorithm,




(a) Time comparison between CPU and FPGA.




(b) Time benchmarks with the array of 20 $\times$ 20.

Fig. 7. (a) Comparison of QRM execution time between CPU and FPGA, when scaling the size of atom array from 10 to 90 with a step size of 20. (b) Execution time comparison of different rearrangement algorithms with the initial array of 20 $\times$ 20. The y-axis for both (a) and (b) is displayed on a logarithmic scale to accommodate the wide range of execution times.

which is only indirectly influenced by the target size. Since we check and move atoms round by round towards the center, several iterations are necessary to completely fill the target area. Consequently, the number of movements needed increases with the target size. In our experiment, four iterations were used to complete the entire process, ensuring successful rearrangement.

We also compare the execution time of our design with other rearrangement approaches [6], [8], [9], as shown in Fig.7(b). The comparative data are taken from the respective original papers. In this benchmark, a fixed array size of 20 $\times$ 20 is

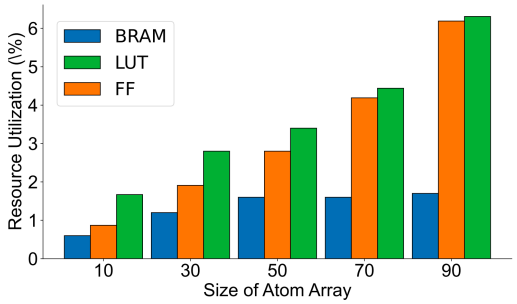



Fig. 8. FPGA resource utilization (percentage) when varying the size of atom arrays.

used, following the experiment setup of works [8], [9]. The Tetris algorithm [6] is executed on an ARM core on an FPGA platform, while the other approaches are all CPU-based. As shown in the figure, our implementations (both on CPU and FPGA) demonstrate significant acceleration over the existing designs. More specifically, benefiting from advanced algorithm design, our CPU-based QRM achieves an acceleration of around 20× over Tetris, 246× over PSCA [9], and almost 1000× over MTA1 [8]. After being implemented on the FPGA, the performance is further improved. Compared with the best candidate (Tetris) in related works, our FPGA-based QRM achieves an acceleration of 120× with only 0.9$\mu s$. Furthermore, attributed to our good scalability, for a larger testing array with 50×50, the acceleration can go up to 300×.

### C. Hardware Utilization

Hardware resource utilization is an important metric to evaluate our design's scalability on the FPGA. Fig. 8 depicts the variation in resource utilization (percentage) with the same test scenario depicted in Fig. 7(a). BRAM exhibits consistent behavior across different array sizes, maintaining stable consumption levels regardless of the array size ranging from 30 to 90. Both LUT and FF utilization demonstrate a linear increasing trend, with FF increasing slightly faster than LUT. However, even with a 90×90 initial array, the LUT consumption remains at 6.31%, and FF consumption at 6.19%, ensuring enough space for other essential functional blocks required for neutral atom quantum computation or even other control logics for other physical modalities (cross-technology control scheme). More specifically, only about half of the resources are occupied by the four QPM, and the other half belongs to the logic to integrate the outputs, which can be optimized in future work.

In summary, the collective analysis of the results presented in Fig. 7(a) and Fig. 8 illustrates the great scalability of our design. It exhibits high adaptability in terms of both latency and hardware performance, effectively accommodating variations in the atom array size.

## VI. Conclusion

In this work, we presented an FPGA-based atom rearrangement algorithm named QRM. The main effort of this work focused on an efficient moving schedule development on the algorithm side and an efficient architecture design on the hardware side. This hardware-software codesign approach significantly reduces the analysis time of atom rearrangement and guarantees a lower clock cycle of neutral atom quantum computers. We evaluated our work with the input atom array of 50 × 50 on a Zynq RFSoC FPGA, and our implementation can finish the rearrangement schedule analysis with around 1.0 $\mu s$. The result indicates a 54× improvement compared to a CPU implementation, and 300× speedup compared to state-of-the-art FPGA work (Tetris algorithm). Furthermore, we tested the execution time and hardware consumption across different sizes of input atom arrays, revealing great scalability, low analysis latency, and efficient hardware utilization.

## References

[1] S. Maurya, C. N. Mude, W. D. Oliver, B. Lienhard, and S. Tannu, "Scaling qubit readout with hardware efficient machine learning architectures," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589042

[2] X. Guo, K. Qin, and M. Schulz, "Hisep-q: A highly scalable and efficient quantum control processor for superconducting qubits," in *2023 IEEE 41st International Conference on Computer Design (ICCD)*, 2023, pp. 86–93.

[3] D. Ohl de Mello, D. Schäffner, J. Werkmann, T. Preuschoff, L. Kohfahl, M. Schlosser, and G. Birkl, "Defect-free assembly of 2d clusters of more than 100 single-atom quantum systems," *Phys. Rev. Lett.*, vol. 122, p. 203601, May 2019. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.122.203601

[4] N. Schlosser, G. Reymond, and P. Grangier, "Collisional blockade in microscopic optical dipole traps," *Phys. Rev. Lett.*, vol. 89, p. 023005, Jun 2002. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.89.023005

[5] M. Morgado and S. Whitlock, "Quantum simulation and computing with rydberg-interacting qubits," *AVS Quantum Science*, vol. 3, no. 2, 2021.

[6] S. Wang, W. Zhang, T. Zhang, S. Mei, Y. Wang, J. Hu, and W. Chen, "Accelerating the assembly of defect-free atomic arrays with maximum parallelisms," *Phys. Rev. Appl.*, vol. 19, p. 054032, May 2023. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevApplied.19.054032

[7] X. Wu, X. Liang, Y. Tian, F. Yang, C. Chen, Y.-C. Liu, M. K. Tey, and L. You, "A concise review of rydberg atom based quantum computation and quantum simulation," *Chinese Physics B*, vol. 30, no. 2, p. 020305, 2021.

[8] S. Ebadi, T. T. Wang, H. Levine, A. Keesling, G. Semeghini, A. Omran, D. Bluvstein, R. Samajdar, H. Pichler, W. W. Ho *et al.*, "Quantum phases of matter on a 256-atom programmable quantum simulator," *Nature*, vol. 595, no. 7866, pp. 227–232, 2021.

[9] W. Tian, W. J. Wee, A. Qu, B. J. M. Lim, P. R. Datla, V. P. W. Koh, and H. Loh, "Parallel assembly of arbitrary defect-free atom arrays with a multitweezer algorithm," *Phys. Rev. Appl.*, vol. 19, p. 034048, Mar 2023. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevApplied.19.034048

[10] J. I. I. Association, "Coaxpress standard gains endorsement from the aia and emva to achieve global standardization status," https://www.coaxpress.com/wp-content/uploads/2016/05/CoaXPress-Official-World-Standard_30Mar11.pdf.

[11] D. Jaksch, J. I. Cirac, P. Zoller, S. L. Rolston, R. Côté, and M. D. Lukin, "Fast quantum gates for neutral atoms," *Phys. Rev. Lett.*, vol. 85, pp. 2208–2211, Sep 2000. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.85.2208

[12] T. Graham, M. Kwon, B. Grinkemeyer, Z. Marra, X. Jiang, M. Lichtman, Y. Sun, M. Ebert, and M. Saffman, "Rydberg-mediated entanglement in a two-dimensional neutral atom qubit array," *Physical review letters*, vol. 123, no. 23, p. 230501, 2019.

[13] D. Barredo, S. de Léséleuc, V. Lienhard, T. Lahaye, and A. Browaeys, "An atom-by-atom assembler of defect-free arbitrary two-dimensional atomic arrays," *Science*, vol. 354, no. 6315, pp. 1021–1023, 2016. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aah3778

[14] AMD, "Zcu216 evaluation board user guide (ug1390)," https://docs.amd.com/r/en-US/ug1390-zcu216-eval-bd/Introduction, 2023.