LSQCA: Resource-Efficient Load/Store Architecture for Limited-Scale Fault-Tolerant Quantum Computing

Takumi Kobori†, Yasunari Suzuki‡, Yosuke Ueno*, Teruo Tanimoto**, Synge Todo†, and Yuuki Tokunaga‡
The University of Tokyo†, NTT Corporation‡, RIKEN*, Kyushu University**
takumi.kobori@phys.s.u-tokyo.ac.jp, yasunari.suzuki@ntt.com, yosuke.ueno@riken.jp,
tteruo@kyudai.jp, wistaria@phys.s.u-tokyo.ac.jp, yuuki.tokunaga@ntt.com

2025 IEEE International Symposium on High Performance Computer Architecture (HPCA) | 979-8-3315-0647-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/HPCA61900.2025.00033

***Abstract*—Current fault-tolerant quantum computer (FTQC) architectures utilize several encoding techniques to enable reliable logical operations with restricted qubit connectivity. However, such logical operations demand additional memory overhead to ensure fault tolerance. Since the main obstacle to practical quantum computing is the limited qubit count, our primary mission is to design floorplans that can reduce memory overhead without compromising computational capability. Despite extensive efforts to explore FTQC architectures, even the current state-of-the-art floorplan strategy devotes 50% of memory space to this overhead, not to data storage, to guarantee unit-time random access to all logical qubits.**

In this paper, we propose an FTQC architecture based on a novel floorplan strategy, Load/Store Quantum Computer Architecture (LSQCA), which can achieve almost 100% memory density. The idea behind our architecture is to separate the whole memory regions into small computational space called Computational Registers (CR) and space-efficient memory space called Scan-Access Memory (SAM). We define an instruction set for these abstract structures and provide concrete designs named point-SAM and line-SAM architectures. With this design, we can improve the memory density by allowing variable-latency memory access while concealing the latency with other bottlenecks. We also propose optimization techniques to exploit properties of quantum programs observed in our static analysis, such as access locality in memory reference timestamps. Our numerical results indicate that LSQCA successfully leverages this idea. In a resource-restricted situation, a specific benchmark shows that we can achieve approximately 90% memory density with 5% increase in the execution time compared to a conventional floorplan, which achieves at most 50% memory density for unit-time random access. Our design is defined as an abstract form, making this principle ubiquitous and applicable to a wide range of quantum devices, qubit-connectivity configurations, and error-correcting codes.

I. Introduction

Fault-tolerant quantum computing (FTQC) is expected to enable faster computation that surpasses classical computers [6], [33], [37], [50], [53], [57], [65]. The major problems in realizing FTQCs are their high error rates and the limited number of available qubits. Quantum error correction (QEC) [56], [60] is a promising solution to reduce error rates by protecting qubits with error-correcting codes. Among the various types of quantum error-correcting codes, surface codes [17], [24], [42] have gained attention due to their high performance and practical implementability. Thus, most FTQC proposals are based on this design [2], [9], [43], [58], [69], and the optimization of quantum algorithms and programs is dedicated to surface-code-based architectures [28], [33], [35], [41], [47], [68]. A drawback of using surface-code-based architecture is the memory overhead required to perform logical operations on the surface-code qubits. Standard logical operations such as lattice surgery and code deformation demand temporal allocations of spatially neighboring surface-code cells as an auxiliary space, which makes the problem of limited qubit counts more serious. Therefore, a strategy to find high-memory-density floorplans for data logical-qubit cells and auxiliary cells is demanded to execute large quantum programs with a small number of qubits, and massive efforts have been paid to find efficient floorplan strategies [7], [8], [22], [44].

However, the existing floorplans have the problem of the resources required for operations increasing proportionally with the number of data cells. Due to this problem, even the state-of-the-art floorplan suffers from low memory density, which can use at most 50% or 67% memory space [8], [44]. This difficulty arises because they aim to maintain parallelism of logical operations as much as possible, ensuring that all logical qubits are immediately ready for operations. From a computer science perspective, this approach resembles a system where all the memory cells are composed of registers, which can be inefficient, especially in a resource-restricted scenario. Additionally, considering the overall computation in FTQC, various potential bottlenecks exist beyond the parallel execution of operations. For example, producing magic states with small error rates is necessary for universal FTQC but is costly in terms of both time and space. Thus, the magic-state generation would be one of the dominant bottleneck factors in resource-restricted FTQC [15], [16], [27], [47]. Therefore, there is a strong demand for designing a floorplan strategy that can reduce the required qubit count with a small penalty of operational parallelism while concealing the overhead of execution time by other bottlenecks.

In this paper, we propose an FTQC architecture based on a novel floorplan strategy called Load/Store Quantum Computer Architecture (LSQCA). This architecture is designed to combine a small computational region and space-efficient memory space. To execute various application programs in this architecture, we propose an instruction set architecture supporting data movement between the two regions as an

979-8-3315-0647-6/25/$31.00 ©2025 IEEE
DOI 10.1109/HPCA61900.2025.00033

abstraction, referring to the relation between the register and memory in conventional computers. For surface-code-based FTQC, we proposed a practical design of LSQCA that consists of Computational Register (CR) as a computational region and Scan-Access Memory (SAM) as a memory space. In this architecture, logical qubits can be loaded from SAM to CR, and logical operations can be performed in CR. Then, the loaded qubits are stored back in SAM. The SAM can achieve nearly 100% memory density by allowing variable-latency memory access while suppressing the increase of average latency by concealing it with other bottlenecks (Fig. 1). The design with CR and SAM also allows flexible tuning of the trade-off between memory overhead and access latency.

LSQCA can be considered a form of memory hierarchy, which can improve average latency by utilizing the access locality in quantum programs. Conventional computers leverage the spatial and temporal access locality in programs by the hierarchical memory subsystem [54]. This strategy ensures the scalability of memory capacity while suppressing average access latency. We performed static analysis on memory reference patterns for FTQC algorithms and found that they have access locality. Then, we integrate locality-aware data movement strategies, and our numerical calculations show that the LSQCA can exploit the locality. We also introduce several crucial concepts to optimize LSQCA, such as multi-bank memories, in-memory operations, and hybrid floorplans.

Our contributions are as follows:

- We propose a novel architecture named LSQCA, introducing the concept of register-memory data movements to FTQC floorplan designs.
- We propose an instruction set architecture of LSQCA and provide designs with CR and SAM. We provide two types of designs compatible with surface-code-based FTQC: point-SAM and line-SAM architectures, exhibiting different characteristics in terms of memory efficiency, latency, and the utilization of access localities.
- The performance and the execution time overhead of LSQCA are evaluated by numerical simulations with realistic quantum programs and various configurations. The results show that LSQCA achieves higher memory density than conventional floorplans with a negligible or modest increase in execution time.
- Under a resource-restricted scenario, LSQCA can achieve 87% and 92% memory density at the cost of 6% and 7% increase in execution time for multiplier and SELECT circuits compared to conventional floorplans, respectively, while conventional floorplans demand 50% of memory space to guarantee unit-time access to arbitrary logical qubits.

Since our architecture is defined as an abstracted form, this principle can be applied to a wide range of qubit devices, connectivity configurations, logical operations methods, and error-correcting codes. In future work, we expect that a more sophisticated instruction scheduler to handle variable memory-access latencies introduced by the LSQCA can further minimize the memory access overhead. For example, variable memory access latencies might be efficiently concealed by leveraging techniques commonly employed in conventional computing, such as strategic data allocation or prefetching based on access pattern detection.

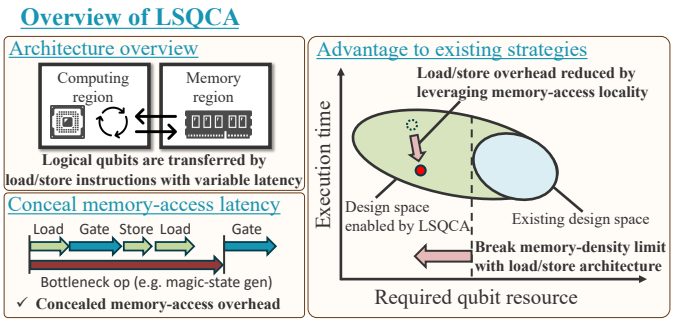


Fig. 1: The positioning of our proposal, LSQCA, which breaks the memory density limit of conventional floorplans and enables the exploration of the area in the trade-off between required resources and execution time where conventional floorplans cannot reach.

II. Background

This section explains the basic part of FTQC. Due to the page limit, we focus on crucial points in our paper. For details on basic theories, please refer to Refs. [28], [30], [47], [53].

A. Advantage of Quantum Computing

Quantum computing is expected to process several scientific problems exponentially faster than those based on classical mechanics [53]. There are several difficulties in developing quantum computers. The main issue is the high error rate of qubits. This problem can be solved by QEC techniques described later, but QEC imposes significant overhead on qubit count and latency. A typical QEC strategy demands hundreds of qubits for constructing a single reliable logical qubit, and the latency for logical operation takes a few tens of microseconds. Additionally, the available number of qubits is limited due to the difficulties in integrating quantum devices. This situation motivates us to find application domains where we can easily demonstrate the advantage of quantum computing and establish compilation methods for them with a few qubits. Currently, solving integer factoring [33] and analyzing the spectrum of condensed matter or molecules [4], [68] are considered promising targets.

B. Surface Codes

To decrease the error rates of qubits, we need to integrate QEC, i.e., encoding qubits into redundant space to enable error detection and correction during the computation. Surface codes [10], [17], [42] are known as one of the most promising quantum error-correcting codes because their error-correction performance is high and they can be implemented only with the nearest-neighboring interactions between qubits allocated on two-dimensional (2D) grids. Thus, the current standard

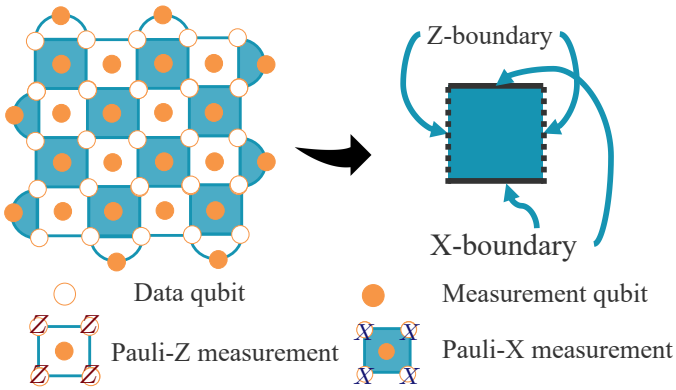


Fig. 2: Illustration of a surface code with distance $d = 5$.

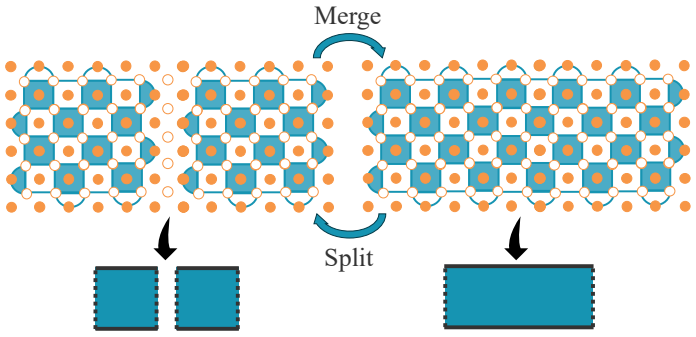


Fig. 3: A procedure of lattice surgery. The merge and split process results in logical Pauli-ZZ measurement.

FTQC architectures are based on the surface codes [20], [25], [31], [40], [64]. A logical qubit with surface codes is represented by physical data qubits placed on a 2D grid, as shown by the white circles in Fig. 2. Errors on data qubits are detected by parity checks on subsets of data qubits. This process is called syndrome measurements. White and blue faces in this figure represent the pattern of syndrome measurements and correspond to two types of measurement, Pauli-Z and -X, on the data qubits at the vertices of the face, respectively. A measurement qubit is placed for each face to enable up to four-qubit Pauli measurements with nearest-neighboring operations, which are shown as orange circles in the figure. During computation, errors can occur not only in the data qubits but also in the measurement qubits. Therefore, error locations are identified sequentially based on the outcomes of repeated syndrome measurements. One syndrome measurement cycle is called a code cycle, and one code cycle takes at least about 1μs [29], [40], [63]. The number of physical qubits along one side is called a code distance d, since surface codes can detect any simultaneous errors acting on less than d qubits. In the arrangement shown in the figure, the sides on the left and right (top and bottom) are referred to as Z-(X-)boundary, respectively. This is because Pauli-Z(X) unitary operations along with the Z-(X-)boundary result in a Pauli-Z(X) unitary operation on encoded logical qubits. In the following discussion, we frequently represent a surface-code patch with a blue cell, where the solid and broken lines correspond to X- and Z-boundaries, respectively.

C. Logical Operation on Surface-Code Qubit

To perform an arbitrary quantum computation on FTQCs, a universal operation set for logical qubits is necessary. Some finite operation sets are known to be universal, i.e., they can construct an arbitrary operation on qubits [53]. We focus on the following universal set since they are popular in constructing FTQCs [28]; three state-preparation operations (zero-state $|0\rangle$, plus-state $|+\rangle$, magic-state $|A\rangle$), five unitary operations (Pauli-X Y Z operations, Hadamard operation H, phase operation S), and four Pauli measurement operations (one-qubit Pauli measurement M_X M_Z, and two-qubit Pauli measurement M_{XX} M_{ZZ}). Among these operations, magic-state preparation, Hadamard operation, phase operation, and two-qubit Pauli measurements have non-negligible latency, and the latency of the other operations can be neglected.

Code deformation and lattice surgery are crucial concepts for enabling the above operations only with the nearest neighboring operations while maintaining fault tolerance [28], [47]. They are realized by modifying the syndrome measurement patterns of surface codes. If two X- and Z-boundaries are separated by $d-1$ blue and white faces, respectively, they can also store one-qubit information with code distance d. During the computation, we can fault-tolerantly change the syndrome measurement patterns, and some changing patterns are known to effectively result in logical operations on qubits.

For instance, lattice surgery can be used for performing two-body logical Pauli measurements [41], [66]. Suppose that two surface-code qubits are independently error-corrected. At the beginning of performing lattice surgery, a new set of parity-check faces are added using qubits between the cells to connect the two Z-(X-)boundaries. After repeating the new set of syndrome measurements for a while, the code returns to the original set. This procedure effectively acts on two logical qubits as logical Pauli-ZZ(-XX) measurements. Fig. 3 illustrates the procedure of lattice surgery with physical and simplified views. If there is a path of unused qubits, we can perform lattice surgery operations in the same manner. To maintain the code distance d against errors, d repeats of syndrome measurements are necessary for reliable error estimation after changing syndrome measurement patterns. Thus, it is convenient to define a d code-cycle period as one code beat, which is used as the basic unit of computing time. In practical applications, the code distance is expected to be between 11 and 31, so one code beat is approximately 10μs to 50μs. The latency of lattice surgery is one code beat.

Logical H and S operations can also be realized similarly by switching syndrome measurement patterns several times [8], [18], [28], [47]. H and S operations temporally demand an additional surface-code cell and take three and two code beats, respectively. The preparation of magic states is performed via a specialized procedure called magic-state distillation [15], [16], [36] and is known to be one of the bottlenecks in FTQC. The generation of magic states is

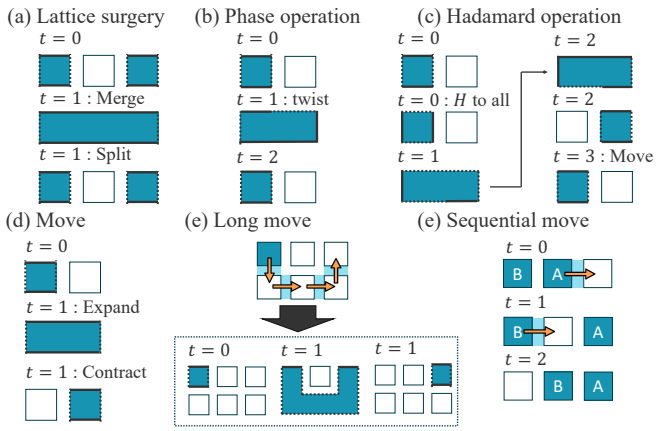


Fig. 4: (a, b, c) Summary of logical operations. (d, e, f) Move operations. t represents the elapsed code beats.

typically performed in a dedicated space called magic-state factories (MSFs). Code deformation can be used for expanding and reducing the patch size of logical qubits without changing the state of logical qubits, and we can move the position of a logical qubit by combining them if there is a path [28], [47]. The time and process required for these primitive surface-code operations are listed in Fig. 4. A notable feature of FTQC based on this strategy is the need for auxiliary surface-code cells to perform primitive FTQC operations.

D. Major Applications and Components

Since the advantage of quantum computers is supported by complexity-theoretic speed-up, the evaluation and optimization for FTQC architectures should be focused on typical and bottleneck structures of quantum algorithms for various application targets rather than general programs [5], [8]. Major FTQC applications can fall into two classes. The first class is the analysis of quantum materials, where material structures are typically provided as a matrix called Hamiltonian represented by a linear combination of Pauli operators, i.e., $H = \sum_{i=0}^{L-1} \alpha_i P_i$ with positive coefficients α_i and Pauli operators P_i. By using a set of techniques known as quantum signal processing and linear combination of unitaries [49], most analysis tasks for Hamiltonian, such as calculating ground energy, simulating dynamics, and other transformations, can be performed by two unitary operations with the following actions [51].

- SELECT: $U_S(\sum_{i=0}^{L-1} |i\rangle |\ \ _i\rangle) = \sum_{i=0}^{L-1} |i\rangle (P_i |\ \ _i\rangle)$
- PREPARE: $U_P |0\rangle = \sum_{i=0}^{L-1} \sqrt{\frac{\alpha_i}{\sum_j \alpha_j}} |i\rangle$

Since these components can be implemented with fewer magic states compared to other approaches such as Trotter-expansions [68], the existing resource estimation and FTQC compilation focus on efficient synthesis and scheduling of the above two operations [4], [8], [44], [68]. The ratio of execution times for these components is calculated in Fig.S11 of Ref. [68], and SELECT would dominate more than 80% runtime of the whole execution when considering instances of condensed matter Hamiltonian with quantum advantages.

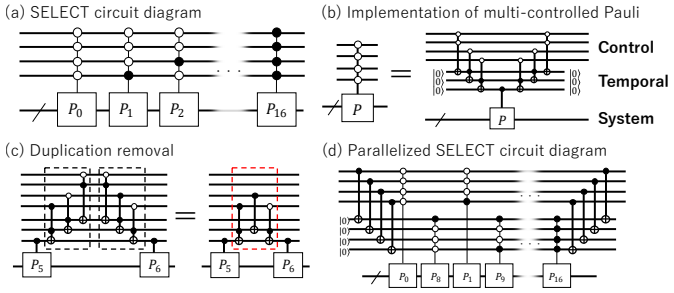


Fig. 5: Circuit diagrams and optimizations of SELECT.

A naive implementation of the SELECT operation can be constructed as an iteration of multi-qubit-controlled Pauli gate $T_i = |i\rangle\langle i| \otimes P_i + (I - |i\rangle\langle i|) \otimes I$ for $0 \leq i < L$, as shown in Fig. 5a. Each T_i can be implemented with a ladder of Toffoli gates, and qubits are classified into three registers: control, temporal, and system (Fig. 5b). The count and depth of Toffoli gates can be further reduced. Babbush *et al.* [4] showed that some Toffoli gates are canceled out, as shown in Fig. 5c. Also, each Toffoli gate in SELECT operations can be decomposed into fewer T-gates than general cases. Yosihoka *et al.* [68] and Boyd [13] showed that the depth of Toffoli gates can be reduced by creating copies of k control registers, i.e., $\sum_{i=0}^{L-1} |i\rangle |\ \ _i\rangle \mapsto \sum_{i=0}^{L-1} |i\rangle^{\otimes k} |\ \ _i\rangle$, and applying Toffoli gates in parallel (Fig. 5d). While further fine-tunes are available, see these references for more detailed optimizations [4], [13], [68].

The other and longer-term application fields are solving integer problems, such as factoring, where problems are provided as integers or equations. A major difficulty in implementing this class of applications is implementing non-Clifford integer-arithmetic circuits, such as integer addition and multiplication. In Sec. III-B, we focus on typical examples of these two dominant application fields, SELECT and integer-multiply circuits.

III. Motivation

Since logical qubit operations require additional empty space, strategies for mapping and control of logical qubits affect the performance and memory density of FTQC architectures. As explained later, all the existing strategies request that all the logical operations can be executed within constant code beats at the cost of low memory density. The purpose of this paper is to show high memory-density architecture based on load/store architecture, where the positions of logical qubits are managed by high-memory-density regions and computation is performed at a dedicated small space. While this choice imposes a penalty of longer and variable latency for memory access, we numerically show that these latencies would be highly likely concealed by other bottleneck factors in popular quantum applications. In this section, we show evidence that the above direction is reasonable; we review the existing floorplans and mapping strategies in Sec. III-A and

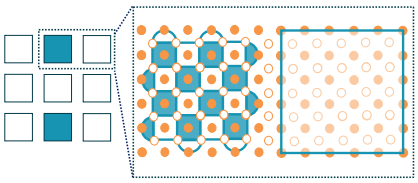


Fig. 6: Diagram of the abstract surface-code in 2D grids. Blue and white cells represent data and auxiliary cells, respectively.

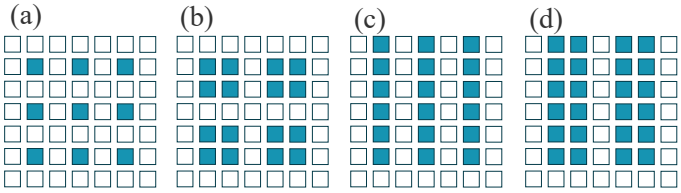


Fig. 7: Floorplans proposed in the existing works. (a) 1/4-filling floorplan [7]. (b) 4/9-filling floorplan [22]. (c) 1/2-filling floorplan [8]. (d) 2/3-filling floorplan [44].

show that static analysis on benchmark programs indicates that fast random access is not demanded in practice in Sec. III-B.

A. Existing Floorplan Strategies

When using multiple logical qubits, it is essential to determine where on the chip to allocate logical qubits auxiliary areas for logical operations. Suppose that the entire set of qubits is divided into surface-code-shaped cells arranged in a 2D grid as shown in Fig. 6, and each cell is used for data cells to store qubit information or as auxiliary cells to assist logical operations. Optimization of this arrangement is equal to finding efficient floorplans, i.e., finding allocations of data and auxiliary cells to maximize the performance of FTQCs. The exploration of efficient floorplans is crucial for solving large problems with a shorter execution time and fewer qubits.

Several floorplan patterns have been investigated in the existing works [7], [8], [22], [28], [44], [47]. The most popular ones are listed in Fig. 7. In these floorplans, 1/4, 4/9, 1/2, and 2/3 of the whole cells are used for data cells. A common feature of these floorplans is that all the data cells have at least one neighboring auxiliary cell, and any pair of data cells has a path of auxiliary cells between them. This feature guarantees that any logical operation on any target data cells can be executed with a constant latency.

The first two floorplans guarantee that any operation can be executed without overheads for allocating auxiliary cells. In the 1/2-filling floorplan, either of X- or Z-boundaries is not exposed to an empty cell and cannot perform an arbitrary lattice surgery in a single code beat. This problem can be mitigated by utilizing a compact form [8], [47], where two logical qubits are constructed by two neighboring patches, and both X- and Z-boundaries are exposed to empty-cell lines, while it induces a small penalty in separately controlling them. The 2/3-filling floorplan cannot allow one-code-beat access to logical qubits since only one of the X- or Z-boundaries of the data cells is exposed to auxiliary cells. Nevertheless, arbitrary operations can be executed with constant code beats in this floorplan. Note that floorplans with a higher memory density are not necessarily superior to those with a lower memory density. A lattice-surgery operation on distant cells in the 1/2 and 2/3 floorplans will consume many auxiliary cells, which block the parallel execution of logical operations. In summary, in the existing proposals, at least 50% of memory cells are devoted to achieving unit-time access to the logical qubits, and at least 33% for constant-time access.

B. Long Memory-Access Latency Tolerance

As explained in the previous section, about half of the integrated qubits are currently used for auxiliary cells to assist logical operations, not data storage, to guarantee unit-time random access to logical qubits. Meanwhile, the small number of available qubits is a major problem in quantum devices. This situation naturally leads to the following question; *Is such fast memory access required in practice?* To address this issue, we prepared benchmarks and analyzed major application components discussed in Sec. II-D, i.e., SELECT and multiplier circuits. Then, we found that, at least in resource-restricted scenarios, fast random access memory would not improve the execution time of typical quantum programs.

For benchmarking with SELECT, we have synthesized the circuits for a 10×10 2D Heisenberg model. For the integer-multiplying circuits, we used the multiplying circuit with 400 logical qubits developed in QASM benchmark [45] and translated it into universal sets of FTQCs. We simulated these circuits with our FTQC simulators and analyzed the memory reference trace. Here, we assumed that magic states are instantly prepared (i.e., there are sufficiently many MSFs), and logical operations can be executed in parallel if their instruction targets do not overlap.

We plotted the reference timestamp of SELECT and integer-multiplying circuits for each logical qubit in Figs. 8a and 8c, and we also plotted the cumulative distribution of reference periods in Figs. 8b and 8d, respectively. For the SELECT circuit, the reference timestamps and distributions of the control, system, and temporal registers are plotted in red, blue, and green, respectively. Since the execution time of the integer-multiplying circuit is long, we plot the first 20,000 code beats among the whole trace. In these traces, the SELECT and integer-multiplying circuits demand a magic state in each 11.6 and 2.14 code beats on average, respectively.

Temporal locality: In both examples, cumulative distributions imply that there are many references with small periods and a few long periods between instructions. This means they have strong temporal locality, and logical qubits can be stored in space-efficient and long-latency memories during such periods with no reference for a long time.

Spatial locality: We observed clear sequential access patterns in the reference timestamps of the integer-multiplying circuits and each of the three registers of SELECT circuits. The observed patterns reflect the structure of the two programs. The SELECT operation is a sequential iteration of integer-value comparisons, and target condensed matter systems have

local interactions. Typical integer-arithmetic circuits iterate bits from the lowest one to the highest one. Thus, even if the random access to the logical memory is slow, this penalty can be concealed by supporting fast sequential accesses.

Average access frequency: The reference timestamps show these two benchmarks have different access frequency patterns. In the SELECT circuits, a few logical qubits in the control and temporal registers are referred to much more frequently than those in the system register. In contrast, the integer-multiplying circuit has almost uniform access frequencies. This implies that, in the case of SELECT circuits, the execution time can be significantly reduced by putting a small portion of logical qubits into fast-access memory regions.

Magic-state consumption: Even if we utilize 176 logical cells for an MSF, one magic state is generated in every 15 code beats [47]. Thus, both circuits demand magic states more frequently than those generated from a single MSF. Therefore, even if fast memory access is provided, we cannot utilize it since the magic-state preparation becomes a bottleneck when a limited number of MSFs are available.

In summary, we can conclude that, in many cases and periods, unit-time random access does not contribute to reducing the execution time of popular FTQC applications in resource-restricted scenarios. In the case of SELECT circuits, most logical qubits become a target of logical instructions much less frequently than a few limited ones. Thus, fast memory access is not demanded by most logical qubits. In both examples, their references are temporally and spatially localized. Thus, there are possibilities that we can leverage these properties to make memory designs space efficient without penalty by sacrificing the access speed. Also, the consumption rates of magic states are faster than the generation rate of a single MSF. Unless massive factories are available, memory access is not a dominant bottleneck, and fast memories cannot speed up the execution time. All of these observations indicate that there is a possibility of designing more efficient FTQC architectures by exploiting these properties. These observations might not be surprising since typical algorithms in conventional computers have the same nature, and hierarchical memory structures are well-designed to exploit them. However, to the best of our knowledge, there is no study on aggressively leveraging the access locality distribution of FTQC algorithms. These results motivate us to explore a novel FTQC architecture direction that improves memory density by reducing auxiliary cells and allowing slow memory access.

IV. Load/Store Quantum Computer Architectures

In this paper, we propose an FTQC architecture based on a novel floorplan strategy: Load/Store Quantum Computer Architectures (LSQCA). This architecture is designed to utilize small computational regions and space-efficient memory space and support load/store instructions to move logical qubits between these two regions to leverage the observed properties of quantum programs in Sec. III. In this section, we explain the basic components of LSQCA in Sec. IV-A and provide an instruction set in Sec. IV-B. Then, we explain a concrete design of LSQCA for surface-code-based FTQCs in Sec. IV-C. This section focuses on a simple construction for readability, and the optimization of LSQCA designs is discussed in Sec. V.

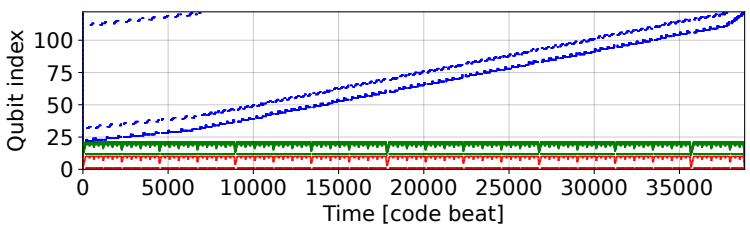
A. Architecture Overview

LSQCA consists of three components: (1) Scan-Access Memory (SAM), (2) Computational Register (CR), and (3) Magic State Factory (MSF) for generating magic states. Each section can be arranged and connected via *ports* as shown in Fig. 9. SAM is a memory region for storing logical qubits. It consists of a large number of logical qubit cells and a small number of auxiliary cells for loading and storing operations on logical qubit cells. The detailed implementation of load/store instructions is explained in the following sections. CR is a computing region for performing logical operations. It is filled by a small number of auxiliary cells. Logical qubits are loaded from SAM to CR, perform logical operations in CR, and then are stored in SAM. MSFs are spaces dedicated to generating distilled magic states. The MSF is connected to CR to supply magic states when it is required. The number of MSFs can be varied to tune the performance, which is called the factory count. Since magic-state preparation can be executed in advance, we can buffer the generated magic states and conceal the latency [38]. Thus, the buffer size is another parameter for optimizing performance. The effect of these parameters on the performance is evaluated in Sec. VI. Ports consist of surface code cells and face each other to transfer logical qubits using lattice surgery. The appropriate number of cells depends on the type of components. To communicate with MSF, we assume a single cell as a port, and we describe it later for SAM. In addition, since the location of logical qubits changes time by time, a controller of SAM(s) is introduced, which keeps the map of variables (`M`) to the location in SAMs.

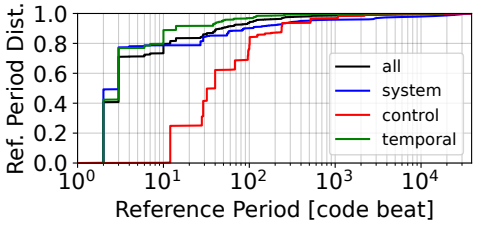
B. LSQCA Instructions

The instructions of the LSQCA are listed in Table I. The LSQCA instructions have three types of operands, memory qubit address (`M`), register qubit identifier (`C`), and classical value identifier (`V`). The memory address and register identifiers specify the abstracted logical cell in SAM and CR, respectively. The classical value identifier is used for measuring the final computational results or storing intermediate logical measurement outcomes and enabling adaptive executions of the following instructions.

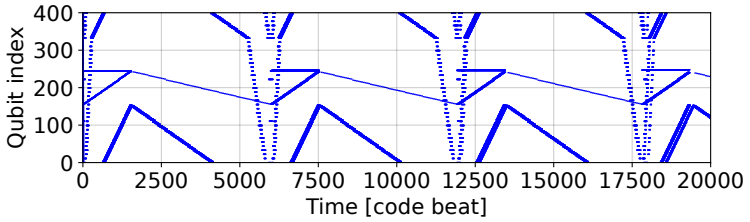
The most characteristic instructions in the LSQCA are `LD` and `ST`, which specify the address or identifier of SAM and CR and move logical qubits between them. These two instructions provide the abstraction to enable LSQCA, where programmers can concentrate on the semantics of the application independently to the logical qubit allocation and the implementation of sophisticated procedures for loading and storing logical qubit cells. They are realized by using several primitive operations to move the positions of logical-qubit cells introduced in Figs. 4 (d, e, f). The instructions categorized into Preparation, Unitary, and Measurement types correspond to logical operations introduced in Sec. II in the CR region.



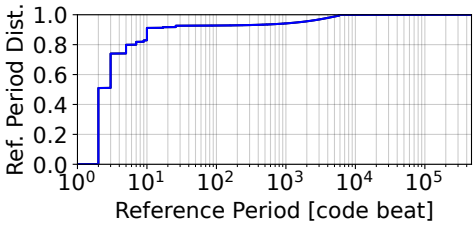
(a) Memory reference timestamp for SELECT



(b) Reference period distribution for SELECT



(c) Memory reference timestamp for multiplier



(d) Reference period distribution for multiplier

Fig. 8: Memory reference pattern analysis for benchmark programs.

TABLE I: LSQCA instructions, where `LD, ST` instructions provide the abstraction for the transportation of logical qubits.

Type	Syntax	Latency	Description
Memory	`LD M C`	variable	(Load) Load logical qubit from SAM to CR
	`ST C M`	variable	(Store) Store logical qubit from CR to SAM
Preparation	`PZ.C C`	0 beat	(Zero-Init) Initialize a logical qubit to $\|0\rangle$ state
	`PP.C C`	0 beat	(Plus-Init) Initialize a logical qubit to $\|+\rangle$ state
	`PM C`	variable	(Magic-init) Move magic state from MSF to CR
Unitary	`HD.C C`	3 beat	(Hadamard) Hadamard gate on a logical qubit
	`PH.C C`	2 beat	(Phase) Phase gate on a logical qubit
Measurement	`MX.C C V`	0 beat	(Pauli-X Meas) Pauli-X measurement on a logical qubit and store outcome
	`MZ.C C V`	0 beat	(Pauli-Z Meas) Pauli-Z measurement on a logical qubit and store outcome
	`MXX.C C1 C2 V`	1 beat	(Pauli-XX Meas) Pauli-XX measurement on logical qubits and store outcome
	`MZZ.C C1 C2 V`	1 beat	(Pauli-ZZ Meas) Pauli-ZZ measurement on logical qubits and store outcome
Control	`SK V`	variable	(Skip) Skip next instruction if a provided value is zero
In-Memory Preparation	`PZ.M M`	0 beat	(Zero-Init) Initialize a logical qubit to $\|0\rangle$ state
	`PP.M M`	0 beat	(Plus-Init) Initialize a logical qubit to $\|+\rangle$ state
In-Memory Unitary	`HD.M M`	variable	(Hadamard) Hadamard gate on a logical qubit
	`PH.M M`	variable	(Phase) Phase gate on a logical qubit
In-Memory Measurement	`MX.M M V`	0 beat	(Pauli-X Meas) Pauli-X measurement on a logical qubit and store outcome
	`MZ.M M V`	0 beat	(Pauli-Z Meas) Pauli-Z measurement on a logical qubit and store outcome
	`MXX.M C M V`	variable	(Pauli-XX Meas) Pauli-XX measurement on logical qubits and store outcome
	`MZZ.M C M V`	variable	(Pauli-ZZ Meas) Pauli-ZZ measurement on logical qubits and store outcome
Optimized Unitary	`CX M1 M2`	variable	(CNOT) CNOT gate on logical qubits

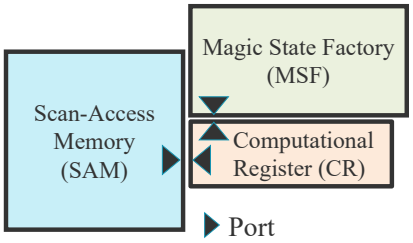


Fig. 9: Conceptual diagram of the LSQCA.

Several standard quantum processes, such as magic-state teleportation or measurement-based quantum computation, need a conditional operation, i.e., performing a logical operation if a preceding measurement value is one. The `SK` operation can be used to implement such adaptive operations. While in-memory computation is not allowed in standard load/store architectures, several in-memory instructions are added to optimize the performance, which is explained later in Sec. V-C. Additionally, we define CNOT gate instruction, which is often used in standard gate sets, as a single instruction, and it can be executed with locally optimized operations to reduce latency. Its detailed explanation is in Sec. VI-A.

Several instructions have variable latency. Memory-type instructions (`LD`, `ST`) have variable latency, since the number of steps required to pick a certain logical cell from SAM depends on their location. The control-type instruction (`SK`) needs to wait for the correction of the target classical value, which depends on the implementation of error estimation algorithms. The magic-state preparation (`PM`) also has a variable latency

since it has to wait for magic-state generation in MSFs when the buffer is empty.

C. Design of CR and SAM

In this section, we explain the detailed design of a core component in our architecture, Computational Register (CR) and Scan-Access Memory (SAM). We propose two types of memory-efficient implementations for SAM: point SAM and line SAM, which have different characteristics in terms of memory efficiency and access latency. Point SAM aims to maximize memory efficiency, whereas line SAM focuses on access latency improvement by exploiting the nature of logical operations on surface code and spatial locality of logical qubit references. We will explain these methods in detail and analyze their characteristics and memory efficiency. Other possible designs will be explored in Sec. IV-D.

1) CR region: CR consists of two columns of cells, as shown in Fig. 10, and is a region for logical operations where arbitrary operations can be performed immediately. The register cells, orange cells in the figure, are located on the right top and right bottom, which possess logical qubits loaded from SAM. In this work, the number of register cells is fixed to be two. The left column of CR acts as a port to connect to SAM. Fig. 10a shows the most compact configuration with six cells.

Since it corresponds to the register file of microprocessors, the CR size significantly impacts CPI. Though there is room for design space exploration in the CR size, we design CR to be as small as possible while exploiting the potential of SAM in this work. This is to maximize the memory efficiency of surface-code-based FTQC and to figure out the distinct characteristics of the concept of LSQCA. Sec. V-D will cover an extensive design space exploration.

2) Point-SAM Architecture: Point SAM is a design of SAM that contains only a single auxiliary cell to maximize memory efficiency. This auxiliary cell is named a *scan cell* and is used for modifying the location of data cells. Supposing that n be the size of memory registers, the shape of point SAM is $\sqrt{n+1} \times \sqrt{n+1}$.[1] In the initial state, the scan cell is set to the nearest cell to the CR region in the center line, as shown in Fig. 10a. The port of point SAM consists of the cells neighboring the initial position of the scan cell, which can also possess logical qubits.

The point SAM loads a target cell in a way similar to sliding puzzles. Using a set of move operations with the scan cell, the target cell can be moved to a diagonal direction with 6 beats and to a horizontal/vertical direction with 5 beats (see Figs. 11 (a, b)). As shown in Fig. 11c, we can load the target cell (green cell) to the CR region by repeating these moves. First, the scan cell is moved to one of the neighboring cells of the target data cell. Then, by repeating the move operations, we can move the position of the target cell to one of the cells of the port. Finally, the target cell is moved to one of the cells for the logical qubits in the CR region.

[1]If $n+1$ is not a square number, we adjust the number by reducing the cells in the bottom line.

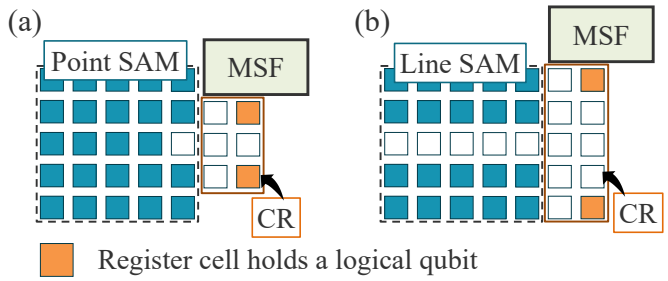


Fig. 10: Schematic diagrams of SAM designs. (a) and (b) shows point-SAM and line-SAM architectures, respectively.

Compared to the conventional floorplans, this architecture achieves asymptotically 100% memory density but does not guarantee unit-time random access to the logical qubits. Suppose that the target cell must be moved W horizontal cells and H vertical cells. Then, the total steps are roughly $W + H + 6\min(W\ H) + 5|W - H|$ beats ignoring a small constant. This takes $7\sqrt{n}$ beats in the worst case, i.e., $W = \sqrt{n}\ H = \sqrt{n}\ 2$.

When one logical cell is loaded from the point SAM, there are two empty cells in the SAM region. Thus, we can use both of them to speed up the load of the second data cell. This technique enables one diagonal move per 4 beats and two vertical/horizontal moves per 6 beats.

The store procedure for a logical-qubit cell to the original position can be performed with the reversed sequence of the load operation. On the other hand, we will utilize another strategy for store instructions to leverage the locality of the memory access. This optimization technique, locality-aware store, will be introduced in Sec. V-B.

3) Line-SAM Architecture: Line SAM is designed with a similar concept to the point SAM, but this uses a dimensionally extended line of auxiliary cells, which is named a scan line, for loading the target logical qubits to CR. The cell allocation of this architecture is shown in Fig. 10b. For line SAM, CR is designed to have the same height as SAM to exploit the nature of logical operations on surface code. In this design, all the cells of line SAM can be regarded as a part of a port. The load operation of the line SAM is performed by vertically moving all the cells in the specific horizontal line that faces the scan line to move it to neighbor the line holding the target cells. Then, the target cell is transported to the CR. This process is illustrated in Fig. 11d.

The latency of the load operations in the line SAM equals the y-axis distance H, which is $0\,5\sqrt{n}$ in the worst case. The store procedure can be performed with its reverse, but we will also introduce a locality-aware strategy for leveraging the locality as explained later in Sec. V-B.

Compared to the point SAM, the line SAM reduces the latency by increasing scan cells. Another feature of the line SAM is the high efficiency of continuous access to data cells on the same line without additional operations for cell movement. In both SAM designs, the memory density be-

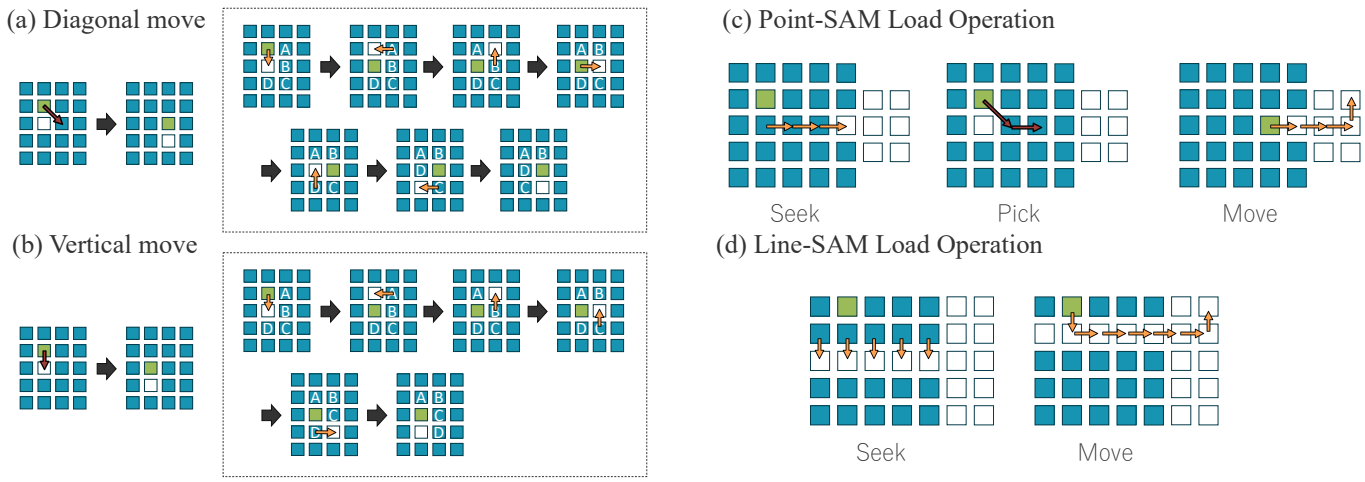


Fig. 11: Procedures to execute Load operation with primitive protocols. (a) and (b) shows a way to move the target cell (green) in diagonal and vertical directions. (c) and (d) shows the load procedure for point-SAM and line-SAM architectures.

comes asymptotically close to 100% as the size of storage n increases.

D. Design space and performance trade-off

While we proposed the designs of CR and SAM in the last section, there are other potential designs in the LSQCA. In this section, we discuss features of SAM/CR that would affect the total performance of LSQCA and their tunability. Based on this discussion, we propose several concrete optimization methods in Sec. V.

In the proposed LSQCA architecture, the size of CR is kept small to maximize the memory density. On the other hand, this direction limits the available instruction-level parallelism (ILP). We can extend the size of CR to allow further ILP at the cost of smaller memory density. This direction will be discussed in Sec. V-D. The bandwidth of SAM can also be improved at the cost of memory density in several ways. One is increasing scan cells in SAM, which improves the average and worst-case latency of memory access at the cost of smaller memory density. In the last section, we proposed line-SAM as intermediates of the existing and point-SAM designs. Another direction is to separate SAM into several banks, which enables parallel access to memory banks and improves memory bandwidth while the memory density per SAM is reduced. This direction is discussed in Sec. V-A.

The above three tunability, CR size, scan cells in SAM, and SAM bank count, trade the memory density with available ILP, memory-access latency, and memory bandwidth, respectively. How large capacity of them we need to ensure is essentially determined by the application structure. Thus, they should be designed according to the analysis of applications or adaptively tuned according to the situation. We show concrete design optimization patterns in the next section and numerically tune several optimization factors for applications in Sec. VI. We believe there are several other ways to design more sophisticated systems, but we left them as future work.

V. Optimizations of LSQCA

While we presented a basic LSQCA in the last section, there is a huge room for improving the performance. In this section, we present four crucial ideas: multiplexing memory access via multi-bank SAM, leveraging reference locality via locality-aware store, reducing the number of load/store instructions by in-memory operations, and hybrid floorplans that use both SAM and conventional floorplans. While there are other possible optimization approaches, more extensive analysis and evaluation of other ideas are left as future work.

A. Multi-Bank SAM

In the proposed SAM designs, we can only access one target cell at a certain time. To break this limit, we can set multiple SAM regions besides the CR region and multiplex the memory access. We named an architecture with this strategy as a multi-bank SAM architecture, and each region is called a bank. Fig. 12a illustrates the allocation of the 2-bank point-SAM and line-SAM architecture, respectively. This enables parallel loading from SAM banks when target data cells are stored in different banks. While there is no essential limit to the number of banks, we need to expand the size of the CR region to let the SAM banks touch the CR. This expansion might degrade the high memory density of the LSQCA, especially in the case of the point-SAM architecture. Thus, we limit the number of banks for point-SAM architecture to two in this paper.

B. Locality-Aware Store

When the most distant data cells are repeatedly accessed, the point- and line-SAM designs impose a huge penalty in the load and store instructions. This is an unavoidable consequence of the trade-off relations between memory density and access latency when attempting to deal with arbitrary access patterns. On the other hand, conventional computers have overcome this limitation by utilizing the spatial and temporal locality inherent in most programs. Fortunately, such locality is also observed

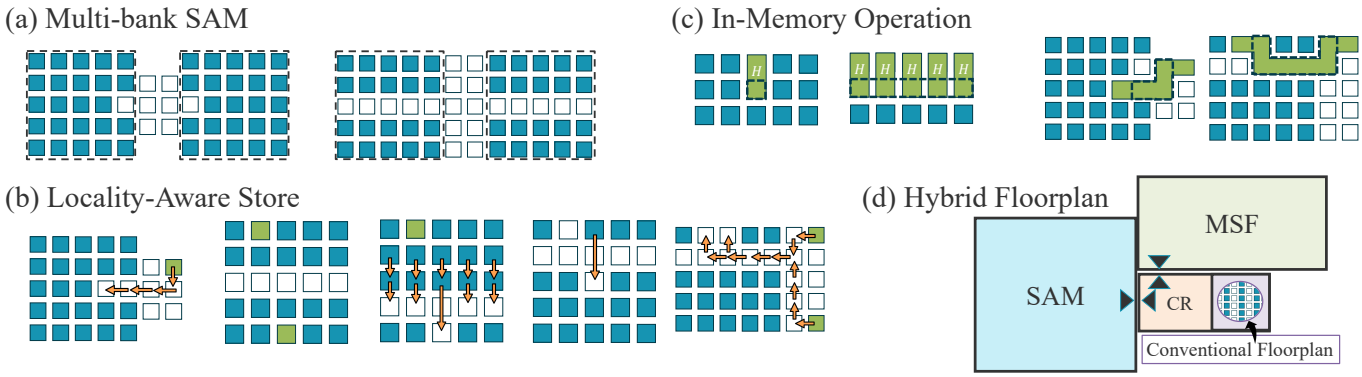


Fig. 12: Optimization methods for the LSQCA.

in practical quantum programs in Sec. III. Thus, we can expect that similar ideas can be employed for the LSQCA.

We can leverage temporal locality in point SAM as follows. Instead of storing the data cell in its original location, the data cells are located in an empty cell near the CR (see the left-most figure of Fig. 12b). This modification not only reduces the beats for store instruction but also makes the distance from the CR region to the cell small. After executing several instructions, frequently accessed cells would typically reside closer regions to the CR, automatically forming efficient cell allocation where the access latency of recently used data becomes small. Thus, this technique can exploit the temporal locality inherent in quantum programs and can significantly reduce the average latency.

We can also modify the line SAM to leverage the spatial locality by refining the store procedure when two data cells are sequentially stored in the same bank. If the reference patterns of quantum programs typically have spatial locality, we can expect a set of logical qubits on which a multi-qubit logical operation acts would be spatially close. Since the line SAM can quickly load the data cells in the same line, such data cells should be stored in the same or neighboring lines. In the locality-aware store for the line SAM, when scanning multiple logical qubit cells for multi-qubit operations, the empty cells in which logical qubits are stored are moved to align with the auxiliary line. Then, two data cells are stored in the same line (see the right four figures of Fig. 12b).

It should be noted that this technique can be introduced without knowledge of quantum programs. Typical optimization of code deformation and lattice surgery patterns needs heavy optimization in the compilation phase, but well-tuned scheduling for a specific FTQC size would lose the portability of programs and is difficult to adapt to latency fluctuations due to probabilistic processes and error-property variations [62]. Therefore, this approach has an advantage that cannot be achieved with compilation techniques.

C. In-Memory Operations

The basic LSQCA assumes every target data cell must be loaded to the CR region before operations. On the other hand, in many cases, the operation can be completed without loading everything to the CR region by using the scan cell or line as auxiliary cells. Therefore, while in-memory instructions are not defined in standard load/store architecture, we define instructions for in-memory operations as a method of LSQCA optimization since this is a natural extension of surface-code-based FTQC. In-memory instructions that can accept SAM address (`M`) are listed at the bottom of Table I.

Several single-qubit logical operations (`PZ`, `PX`, `MX`, `MZ`) do not need auxiliary cells, and they can be executed without moving the scan cell. The other single-qubit logical operations need a single auxiliary cell as described in Fig. 4. Thus, when the scan cell/line reaches the appropriate neighboring cells of the target cell, we can perform logical operations in the SAM. The effect of this technique is significant especially in the case of point SAM, because we can skip the procedures to pick the target data cells to the CR region. Additionally, in the line-SAM architecture, it is possible to simultaneously apply H and S to data cells along the line using a line of auxiliary cells as shown in the left two figures of Fig. 12c. [2]

We can apply the same principle to instructions acting on two data cells, i.e., lattice surgery. As shown in the right two figures of Fig. 12c, when the target data has a path of auxiliary cells to CR, we can directly perform lattice surgery operations between the two target data cells. This will remove the last and first move of the load and store procedure, respectively.

Note that in-memory operation techniques reduce the benefit of the locality-aware store since this technique will skip store instructions. Thus, the choice of in-memory operation or locality-aware store should be examined for each instruction. This point will be discussed in Sec. VI-A.

D. Hybrid Floorplan

Some logical qubits might be heavily accessed during the whole quantum program. In this case, even with the technique to leverage access locality, the latency of load and store instructions would not be negligible. Also, since the ILP is limited by the number of register cells, there is a significant penalty in the execution time if it does not meet the ILP demanded by applications.

[2]While S operations are boundary-sensitive, we can adapt this by rotating cells in advance.

A possible solution to mitigate this problem is to expand the CR regions. To this end, instead of varying the number of cells in CR, we attach a conventional floorplan to effectively increase the number of register cells, as shown in Fig. 12d. By allocating heavily accessed logical qubits to the conventional regions, we can reduce the number of load/store instructions to facilitate faster and more memory-efficient computation and can improve the available ILP. When we use a hybrid floorplan, the size of the computational register identifier ($\mathbb{C}$) is extended according to the area of introduced regions with a conventional floorplan.

While the use of the hybrid floorplan degrades the memory density, we can improve the available ILP and reduce the number of load/store instructions by tuning the size of conventional floorplans while keeping the penalty of the LSQCA modest. The sensitivity of the size of conventional floorplans to the execution time highly depends on the target quantum programs, which is examined in Sec. VI-C.

VI. Performance Evaluation

This section evaluates the performance of the LSQCA in comparison to existing strategies with realistic instances.

A. Numerical Setting

Performance evaluation method: The performance of our architecture is evaluated with Code beats Per Instruction (CPI) and memory density. To this end, we developed a code-beat-accurate simulator that can track the primitive operations decomposed from load/store instructions and can calculate the execution time and CPI, which is the ratio of execution time to the command count of LSQCA. The bank and factory counts are varied in each benchmark. The hybrid floorplan is not used in the benchmark in Sec. VI-B but is used in Sec. VI-C. We always use locality-aware store when the store instructions are executed. In the simulation and evaluation, we utilized the following assumptions. When we encounter conditioned operations, we always choose the taken path, i.e., we always choose the path with more instructions. We ignore instructions with negligible latency, such as Pauli unitary operations, in our evaluation. Our memory density includes SAM banks and CR but excludes MSFs, as their memory density depends on their specific design. We use a popular design of MSFs that can generate one magic state per 15 code beats, proposed by Litinski [47]. The number of magic-state buffers is set to $2n$ for a given number of factories n. The shape of the SAM banks is assumed to be either $L \times L$ or $L \times (L+1)$, and under these conditions, the configuration is set to maximize memory density. Other shapes are considered as future work. When using multiple SAM banks, logical qubits are distributed sequentially to all the banks in order. High-performance assignment of logical qubits to multiple banks is left as future work. For CR, as mentioned in Sec. IV-C, it assumes the smallest unit, and after logical cells are loaded and computations are executed, they are immediately stored. Note that since we count the execution time in the unit of code beats and evaluate memory density in the unit of cells, the performance is independent of the chosen code distance or error rates of physical qubits.

Benchmark program compilation: Each benchmark program is decomposed into the load/store instructions as follows. First, they are represented by Clifford operations (H, S, and CNOT gates), T gates, and single-qubit Pauli measurements. Each T gate is decomposed into Pauli-ZZ measurement on target qubit and magic state, and S gate on target qubit according to the measurement outcome. Then, each gate is translated to our instruction set by attaching load/store instructions to each gate. When we translate single qubit gates, in-memory instructions are always used. For two-qubit gates, we use in-memory operations for the second-operand qubit if the latency for loading the first operand qubit is faster than the other. To manage this at runtime, we defined optimized unitary instruction CNOT in Table. I. Also, we use an in-memory operation for the Pauli-ZZ measurement induced from T gates. The other gates are translated with attaching load and store instructions. We can consider further optimizations for compilation schemes, such as adaptive translation or instruction reordering, which are left as future work.

Baseline setting: Since there are several floorplan strategies in the existing work, we compare the LSQCA performance with that of an optimistic baseline, which we call a conventional floorplan, designed by combining the advantages of several existing floorplans. We set the memory density of the conventional floorplan as 50% from Fig. 7c since this is the maximum density that allows random access with negligible routing overheads. While the ILP is limited by the memory density and qubit locations due to path conflicts of lattice surgery, we assume there are no path conflicts in the conventional floorplan. This means the logical qubit mapping on the conventional floorplan does not affect its performance, which makes evaluation simple. When we consider the hybrid floorplan in Sec. VI-C, we use conventional floorplans for the hybrid floorplan. The above assumption is optimistic for the baseline or equally affects the performance of the proposed architecture and baseline. Thus, this comparison is sufficient to show the advantage of LSQCA over the existing strategies, while this assumption makes evaluation independent from detailed optimization on the baseline, such as logical qubit mapping optimization according to the program structure.

B. Evaluation with Benchmark Programs

First, we show the results of the execution time evaluation for several actual quantum programs. In addition to SELECT and multiplier quantum programs used in Sec. III, we selected five benchmark programs from the QASM benchmark [45]: Quantum adder (adder), Bernstein-Vazirani algorithm (bv), preparing cat state (cat), preparing GHZ state (ghz), and circuits for computing the square root of a number via amplitude amplification (square root). The number of required logical qubits is as follows: 433 for adder, 280 for bv, 260 for cat, 127 for ghz, 400 for multiplier, 60 for square root, and 143 for SELECT circuit for the 11×11 2D Heisenberg model. The results are shown in Fig. 13 with several factory counts.

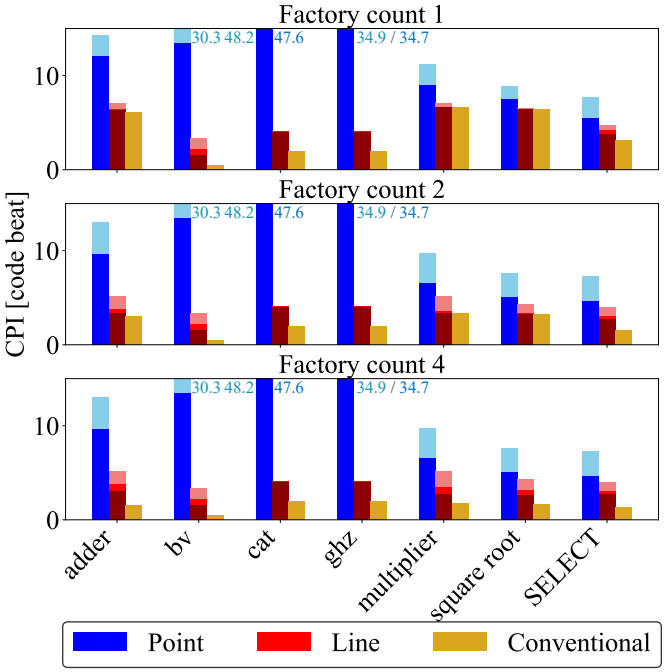


Fig. 13: Performance benchmark results. Each figure shows the CPI with a different number of MSFs.

The blue, red, and yellow bars correspond to the performance of LSQCA with point SAM and line SAM and that with conventional floorplans, respectively. The number of banks is also tested with settings of 1 and 2 for point SAM, and 1, 2, and 4 for line SAM. The darker the color, the greater the number of banks.

The LSQCA successfully improves memory density, which is given as the number of required logical qubits by the application over the total logical qubit count in the architecture. For instance, in the case of a multiplier with only one factory, using Line SAM achieves approximately $400\ 462 \simeq 87\%$ memory density with 6% execution time overhead. The comprehensive results appear in Fig. 14. When the number of factories is limited to one, as assumed in limited-scale FTQC, the comparison between LSQCA and conventional floorplans shows significant differences for bv, cat, and ghz instances, and small differences for adder, multiplier, square root, and SELECT instances. This is a natural consequence because the formers do not demand any magic states, and all gates can be executed without any magic-state generation delay with high parallelism. In this case, the latency of load/store operations is not concealed, resulting in a large overhead in execution time. On the other hand, instructions in the latter circuits demand many magic states and have relatively low parallelism. In such circuits, the penalty of LSQCA is effectively concealed. The use of multiple banks enhances this trend. We note that penalties in the former benchmark (bv, cat, and ghz) are not a serious problem since they are not typically the bottleneck of typical useful quantum algorithms, while the latter examples are heavily repeated bottlenecks in practical algorithms as mentioned in Sec. II-D.

As the number of MSFs increases, the bottleneck caused by the lack of magic states is alleviated. In such a regime, while the execution time of LSQCA and conventional floorplans improves, the discrepancy in the performance also expands. On the other hand, using the multi-bank technique improves the parallelism of LSQCA and can close the performance gap between LSQCA and conventional floorplans in benchmark cases with high parallelism.

Note that when we choose a configuration with a large execution time overhead, the total logical error rate increases, and code distances must be increased, which degrades the memory density improvement. Nevertheless, as far as the execution time penalty is modest, this effect would be negligible in practice. Also, the execution time overhead can be tuned with hybrid floorplan optimization introduced in the next section.

C. Optimization with hybrid floorplan

We perform further optimization using a hybrid layout using the same benchmark programs as in Sec. VI-B and present the results showing the relationship between memory density and execution time. To tune the size of the conventional floorplan in the hybrid layout, we define a ratio of conventional floorplan f. Supposing the number of data cells is n, the size of the conventional floorplan is nf, and that of SAM is $n(1-f)$. We put the most frequently accessed nf data cells into the conventional floorplan and the other into the SAM.

The results are summarized in Fig. 14. In addition to the benchmark results, their GEOMEAN is also shown to evaluate the average performance of LSQCA. The horizontal axis represents memory density, and the vertical axis represents the overhead in execution time compared to the conventional floorplan. For each SAM layout, the ratio f is varied from 0 to 1 by 0.05, and these points are connected with dashed lines. Note that the two endpoints $f = 0$ and $f = 1$ correspond to LSQCA without a hybrid floorplan and the baseline in the previous evaluation, respectively.

In every case, we observed the trade-off relation between memory density and execution time overhead. Compared to the cases of bv, cat, and ghz circuits, the execution time overhead by the hybrid floorplan is modest in the other cases, i.e., adder, multiplier, square root, and SELECT circuits. In the case of the multiplier, square root, and SELECT, the trend of the plots changes at a specific ratio f. In particular, the SELECT circuits show the most drastic trend change at $f = 0\,95$. We expect this is because the control and temporal registers are heavily referenced by many instructions compared to the system register, as observed in Sec. III.

Finally, we evaluated the efficacy of hybrid layouts in terms of instance sizes with the SELECT circuits. We assume the hybrid floorplan in which the data cells for control and temporal registers are placed in the conventional floorplan. The chosen instance sizes (the width of spin lattices of 2D Heisenberg models) are 21, 41, 61, 81, and 101. Their required data cells are 467, 1,711, 3,753, 6,595, and 10,235, respectively. Fig. 15 shows the evaluation results. The memory density increases

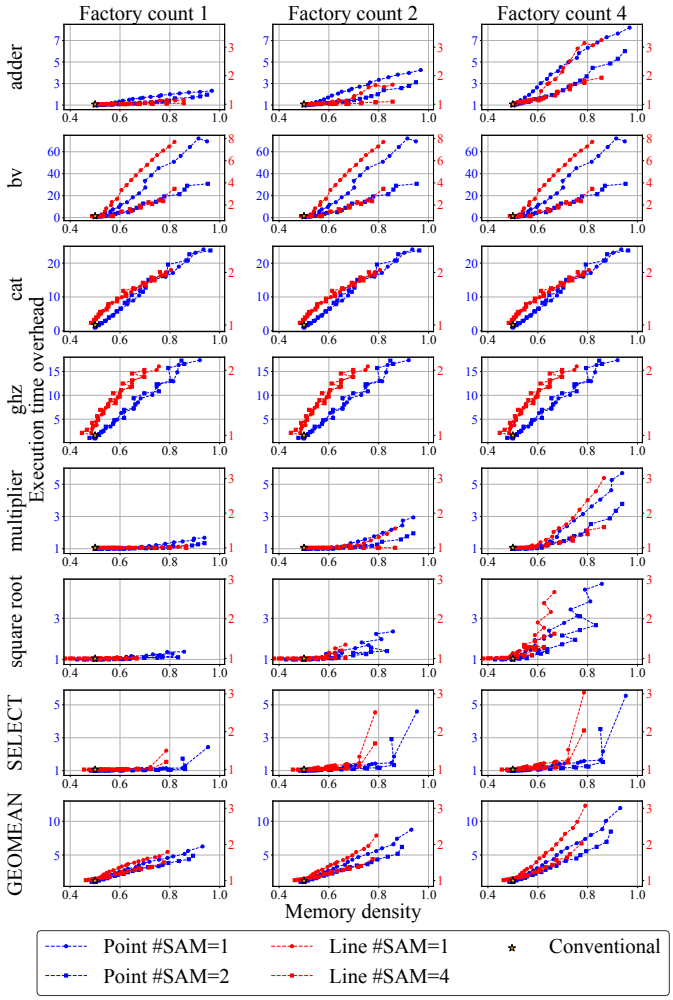


Fig. 14: Performance of LSQCA with hybrid layout. Each figure shows the trade-off between memory density and execution time overhead compared to the conventional floorplan.

as the instance size increases since the CR size becomes negligible. Thus, points are plotted from left to right as the instance size increases. In resource-limited cases (instance size of 21, requiring 467 logical qubits, and with one MSF), Hybrid Point achieves approximately 92% memory density with 7% execution time overhead. In resource-abundant cases (instance size of 101, requiring 10,235 logical qubits, and with four MSFs), Hybrid Line achieves approximately 94% memory density with a 6% execution time overhead. This means that the LSQCA reduces the qubit count for SELECT circuits with a modest penalty not only for small-size instances but also for large ones by introducing hybrid floorplans. This indicates that, when treating large circuits with heavily accessed logical qubits, LSQCA would be an indispensable concept that can significantly improve the computational capability of FTQCs.

VII. Applicable Scope and Related Works

A. Applicable scope of LSQCA

The essential concept of LSQCA is to divide the qubit space into computing space (CR) and high-memory-density space (SAM) at a logical level and improve the memory density while concealing CPI overheads; thus, the concept can be applied to any qubit device, QEC codes, and qubit topology, as far as they are error-corrected and have a logical-level access locality. As a practical and widely applicable design instance of the LSQCA, this paper proposed SAMs and floorplans. This design space is available in architectures with a 2D-grid array of surface-code patches with nearest-neighboring interactions, which is a standard model in many FTQC theories and architectures [8], [22], [28], [47]. Thus, quantum devices, such as superconducting circuits [1], [2], [23], trapped ions [19], [26], [52], and optical photons [11], [12] aim to achieve the same model at a logical level. While some architectures can utilize more flexible connectivity by 3D integration [55], multi-chip modules [59], multi-mode resonator array [25], 2D loop-array [21], or optical tweezer arrays [9], they do not allow constant-time random-access to two-logical-qubit operations unless sacrificing the memory density. For example, neutral atoms can directly move positions, but their shuttling latency increases with physical distances, while lattice surgery can be executed with a constant latency. Thus, while we might need to refurbish the concept for point- and line-SAM, the idea of LSQCA would be versatile to improve memory density by modestly sacrificing CPI in various architectures.

B. Related work

Several existing works successfully improved FTQC architectures by mixing different designs on floorplans. The major existing strategies are to apply different optimization schemes to resource-state generation. Holmes *et al.* [39] proposed a distributed design of MSFs. They extracted algorithm properties such as T-count and magic-state consumption rates from realistic application algorithms and utilized them to maximize the performance of FTQCs. Litinski [47], [48] extensively explored optimized floorplans dedicated to MSFs. Several hand-optimized designs are shown under trade-offs between size and throughput according to the precise analysis of the magic-state fidelities during the distillation process. Stein *et al.* [61] proposed a framework that enables the exploration of heterogeneous designs in broad layers of FTQCs. They demonstrated the flexibility of the framework in the case of superconducting qubits and represented high-level protocols such as error correction, entanglement distillation, and magic-state distillation. Using this framework and leveraging the optimized choice of low-level quantum hardware, they improved the logical error rates for QEC and resource generation protocols. Xu *et al.* [67] and Bravyi *et al.* [14] proposed an idea of using two QEC codes with different properties. The quantum low-density parity check (qLDPC) codes are known to have good encoding rates but are difficult to execute a universal set of logical operations. They proposed qLDPC codes compatible with superconducting circuits and neutral atoms and a method to move logical qubits between qLDPC and surface code regions. Compared to these studies, the significant points of our proposal are as follows.

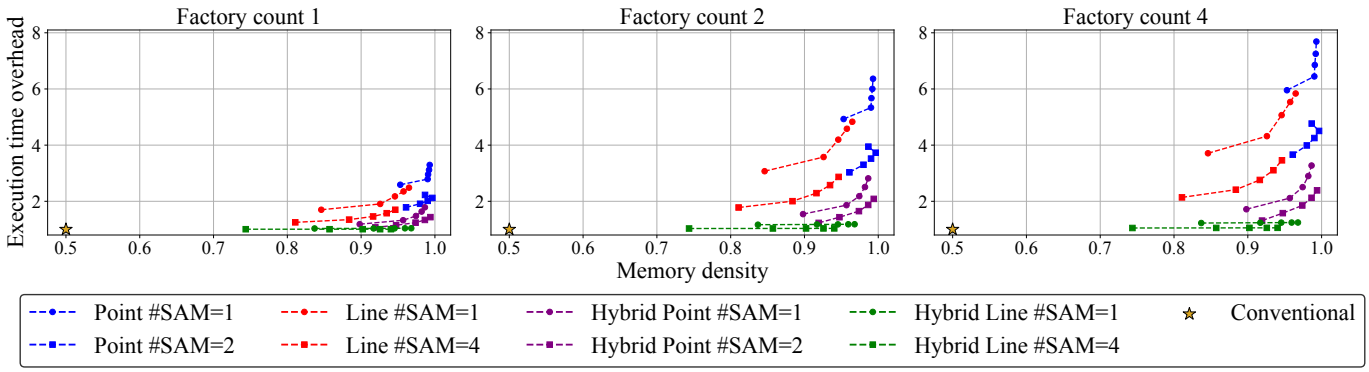


Fig. 15: Performance of the LSQCA for several instance sizes of the SELECT circuits. Each figure shows the trade-off between memory density and execution time overhead compared to the conventional floorplan.

Breaking the existing density limit: The memory density of surface-code-based architectures particular about constant-time access has a limitation to improve. Due to this reason, the proposed floorplan that can be used for general quantum programs is limited to 1/2, or 2/3 with execution overheads. The LSQCA enables the exploration of designs beyond the wall by sacrificing the constant-time access to the logical qubits, based on the observation that such fast access is not demanded in bottleneck components of major applications.

Exploiting application characteristics beyond T-gate: Several existing works have explored the direction of optimizing the FTQC design according to the algorithm structure. Holmes *et al.* [39] exploit the information of magic states and T-gates, such as the count and consumption rate of magic states, and Litinski [47], [48] also exploits the detailed structures and trade-offs in MSF designs. Stein *et al.* [61] proposed the framework to exploit the design space in low-level hardware, such as a heterogeneous combination of superconducting devices. Our paper focuses on exploiting inherent algorithm structures such as access locality and instruction parallelism in addition to T-gate information. Thus, we leveraged high-level information compared to Ref. [61] and more detailed properties of algorithms compared to Refs. [39], [47], [48]. Analyzing these properties and leveraging them for concealing memory access latencies are first achieved in this paper and represent one of the major contributions.

Program Portability: To the best of our knowledge, most existing papers that consider qubit connectivity treat it by using quantum gates on target qubits with explicit physical locations. This means compiled object codes can be executed only for a specific target device, or connectivity needs to be resolved at runtime. While Stein *et al.* [61] also studied abstracted memory, their concepts are motivated by extending design space from choices of multiple devices to high-level FTQC subroutines. Since our paper targets logical-level memory designs and latencies for general computation, the aim of abstraction is different from the existing works. Our paper separates issues of resolving logical-qubit positions from operational instructions and delegates them to abstract SAMs, which enables instruction sets independent of qubit locations. With this abstraction, this paper first provides application-level portability, i.e., common object codes can be executed in every instance of LSQCA in an efficient manner.

Since the LSQCA targets the trade-off between the memory density and CPIs for a given MSF configuration, our proposal is compatible with ideas to optimize MSFs. We also note that while magic-state counts and MSFs have been considered dominant bottlenecks in FTQCs, recent theoretical development significantly reduced the cost of them [3], [4], [8], [32], [34], [44], [46]–[49]. Thus, the design space exploration of FTQCs focusing not only on MSFs and magic-state counts like this proposal would become more vital in the future.

VIII. Conclusion

In this paper, we propose LSQCA, designed to implement a Load/Store architecture on FTQC. This architecture enables efficient storage for rarely-accessed qubits and virtualized addressing for abstract components, SAM banks, CR, and MSFs. We proposed two designs that can be implemented with primitive operations for surface-code-based FTQCs: the point-SAM architecture and the line-SAM architecture. Our numerical evaluation with practical instances indicates that our architecture improves memory density while concealing the penalty of memory access latency, which implies that our architecture is useful for accelerating the demonstration of quantum advantages with FTQCs.

Acknowledgements

This work was supported by JST PRESTO Grant Number JPMJPR2015, JST Moonshot R&D Grant Number JPMJMS2061, JPMJMS2067, and JPMJMS226C, JST CREST Grant Number JPMJCR23I4 and JPMJCR24I4, MEXT Q-LEAP Grant Number JPMXS0120319794 and JPMXS0118068682, the Center of Innovation for Sustainable Quantum AI, JST Grant Number JPMJPF2221, JST SPRING, Grant Number JPMJSP2108, JSPS KAKENHI Grant Numbers 22H05000, 22K17868, and 24K02915, RIKEN Special Postdoctoral Researcher Program.

REFERENCES

[1] R. Acharya, L. Aghababaie-Beni, I. Aleiner, T. I. Andersen, M. Ansmann, F. Arute, K. Arya, A. Asfaw, N. Astrakhantsev, J. Atalaya, R. Babbush, D. Bacon, B. Ballard, J. C. Bardin, J. Bausch, A. Bengtsson, A. Bilmes, S. Blackwell, S. Boixo, G. Bortoli, A. Bourassa, J. Bovaird, L. Brill, M. Broughton, D. A. Browne, B. Buchea, B. B. Buckley, D. A. Buell, T. Burger, B. Burkett, N. Bushnell, A. Cabrera, J. Campero, H.-S. Chang, Y. Chen, Z. Chen, B. Chiaro, D. Chik, C. Chou, J. Claes, A. Y. Cleland, J. Cogan, R. Collins, P. Conner, W. Courtney, A. L. Crook, B. Curtin, S. Das, A. Davies, L. De Lorenzo, D. M. Debroy, S. Demura, M. Devoret, A. Di Paolo, P. Donohoe, I. Drozdov, A. Dunsworth, C. Earle, T. Edlich, A. Eickbusch, A. M. Elbag, M. Elzouka, C. Erickson, L. Faoro, E. Farhi, V. S. Ferreira, L. F. Burgos, E. Forati, A. G. Fowler, B. Foxen, S. Ganjam, G. Garcia, R. Gasca, E. Genois, W. Giang, C. Gidney, D. Gilboa, R. Gosula, A. G. Dau, D. Graumann, A. Greene, J. A. Gross, S. Habegger, J. Hall, M. C. Hamilton, M. Hansen, M. P. Harrigan, S. D. Harrington, F. J. H. Heras, S. Heslin, P. Heu, O. Higgott, G. Hill, J. Hilton, G. Holland, S. Hong, H.-Y. Huang, A. Huff, W. J. Huggins, L. B. Ioffe, S. V. Isakov, J. Iveland, E. Jeffrey, Z. Jiang, C. Jones, S. Jordan, C. Joshi, P. Juhas, D. Kafri, H. Kang, A. H. Karamlou, K. Kechedzhi, J. Kelly, T. Khaire, T. Khattar, M. Khezri, S. Kim, P. V. Klimov, A. R. Klots, B. Kobrin, P. Kohli, A. N. Korotkov, F. Kostritsa, R. Kothari, B. Kozlovskii, J. M. Kreikebaum, V. D. Kurilovich, N. Lacroix, D. Landhuis, T. Lange-Dei, B. W. Langley, P. Laptev, K.-M. Lau, L. L. Guevel, J. Ledford, K. Lee, Y. D. Lensky, S. Leon, B. J. Lester, W. Y. Li, Y. Li, A. T. Lill, W. Liu, W. P. Livingston, A. Locharla, E. Lucero, D. Lundahl, A. Lunt, S. Madhuk, F. D. Malone, A. Maloney, S. Mandrá, L. S. Martin, S. Martin, O. Martin, C. Maxfield, J. R. McClean, M. McEwen, S. Meeks, A. Megrant, X. Mi, K. C. Miao, A. Mieszala, R. Molavi, S. Molina, S. Montazeri, A. Morvan, R. Movassagh, W. Mruczkiewicz, O. Naaman, M. Neeley, C. Neill, A. Nersisyan, H. Neven, M. Newman, J. H. Ng, A. Nguyen, M. Nguyen, C.-H. Ni, T. E. O'Brien, W. D. Oliver, A. Opremcak, K. Ottosson, A. Petukhov, A. Pizzuto, J. Platt, R. Potter, O. Pritchard, L. P. Pryadko, C. Quintana, G. Ramachandran, M. J. Reagor, D. M. Rhodes, G. Roberts, E. Rosenberg, E. Rosenfeld, P. Roushan, N. C. Rubin, N. Saei, D. Sank, K. Sankaragomathi, K. J. Satzinger, H. F. Schurkus, C. Schuster, A. W. Senior, M. J. Shearn, A. Shorter, N. Shutty, V. Shvarts, S. Singh, V. Sivak, J. Skruzny, S. Small, V. Smelyanskiy, W. C. Smith, R. D. Somma, S. Springer, G. Sterling, D. Strain, J. Suchard, A. Szasz, A. Sztein, D. Thor, A. Torres, M. M. Torunbalci, A. Vaishnav, J. Vargas, S. Vdovichev, G. Vidal, B. Villalonga, C. V. Heidweiller, S. Waltman, S. X. Wang, B. Ware, K. Weber, T. White, K. Wong, B. W. K. Woo, C. Xing, Z. J. Yao, P. Yeh, B. Ying, J. Yoo, N. Yosri, G. Young, A. Zalcman, Y. Zhang, N. Zhu, and N. Zobrist, "Quantum error correction below the surface code threshold," *arXiv preprint arXiv:2408.13687*, 2024.

[2] R. Acharya, I. Aleiner, R. Allen, T. I. Andersen, M. Ansmann, F. Arute, K. Arya, A. Asfaw, J. Atalaya, R. Babbush, D. Bacon, J. C. Bardin, J. Basso, A. Bengtsson, S. Boixo, G. Bortoli, A. Bourassa, J. Bovaird, L. Brill, M. Broughton, B. B. Buckley, D. A. Buell, T. Burger, B. Burkett, N. Bushnell, Y. Chen, Z. Chen, B. Chiaro, J. Cogan, R. Collins, P. Conner, W. Courtney, A. L. Crook, B. Curtin, D. M. Debroy, A. Del Toro Barba, S. Demura, A. Dunsworth, D. Eppens, C. Erickson, L. Faoro, E. Farhi, R. Fatemi, L. Flores Burgos, E. Forati, A. G. Fowler, B. Foxen, W. Giang, C. Gidney, D. Gilboa, M. Giustina, A. Grajales Dau, J. A. Gross, S. Habegger, M. C. Hamilton, M. P. Harrigan, S. D. Harrington, O. Higgott, J. Hilton, M. Hoffmann, S. Hong, T. Huang, A. Huff, W. J. Huggins, L. B. Ioffe, S. V. Isakov, J. Iveland, E. Jeffrey, Z. Jiang, C. Jones, P. Juhas, D. Kafri, K. Kechedzhi, J. Kelly, T. Khattar, M. Khezri, M. Kieferová, S. Kim, A. Kitaev, P. V. Klimov, A. R. Klots, A. N. Korotkov, F. Kostritsa, J. M. Kreikebaum, D. Landhuis, P. Laptev, K.-M. Lau, L. Laws, J. Lee, K. Lee, B. J. Lester, A. Lill, W. Liu, A. Locharla, E. Lucero, F. D. Malone, J. Marshall, O. Martin, J. R. McClean, T. McCourt, M. McEwen, A. Megrant, B. Meurer Costa, X. Mi, K. C. Miao, M. Mohseni, S. Montazeri, A. Morvan, E. Mount, W. Mruczkiewicz, O. Naaman, M. Neeley, C. Neill, A. Nersisyan, H. Neven, M. Newman, J. H. Ng, A. Nguyen, M. Nguyen, M. Y. Niu, T. E. O'Brien, A. Opremcak, J. Platt, A. Petukhov, R. Potter, L. P. Pryadko, C. Quintana, P. Roushan, N. C. Rubin, N. Saei, D. Sank, K. Sankaragomathi, K. J. Satzinger, H. F. Schurkus, C. Schuster, M. J. Shearn, A. Shorter, V. Shvarts, J. Skruzny, V. Smelyanskiy, W. C. Smith, G. Sterling, D. Strain, M. Szalay, A. Torres, G. Vidal, B. Villalonga, C. Vollgraff Heidweiller, T. White, C. Xing, Z. J. Yao, P. Yeh, J. Yoo, G. Young, A. Zalcman, Y. Zhang, and N. Zhu, "Suppressing quantum errors by scaling a surface code logical qubit," *Nature*, vol. 614, no. 7949, p. 676–681, Feb. 2023.

[3] Y. Akahoshi, K. Maruyama, H. Oshima, S. Sato, and K. Fujii, "Partially fault-tolerant quantum computing architecture with error-corrected clifford gates and space-time efficient analog rotations," *PRX Quantum*, vol. 5, no. 1, p. 010337, 2024.

[4] R. Babbush, C. Gidney, D. W. Berry, N. Wiebe, J. McClean, A. Paler, A. Fowler, and H. Neven, "Encoding electronic spectra in quantum circuits with linear T complexity," *Physical Review X*, vol. 8, no. 4, p. 041015, 2018.

[5] R. Babbush, J. R. McClean, M. Newman, C. Gidney, S. Boixo, and H. Neven, "Focus beyond quadratic speedups for error-corrected quantum advantage," *PRX quantum*, vol. 2, no. 1, p. 010103, 2021.

[6] B. Bauer, S. Bravyi, M. Motta, and G. K.-L. Chan, "Quantum algorithms for quantum chemistry and quantum materials science," *Chemical Reviews*, vol. 120, no. 22, pp. 12 685–12 717, 2020.

[7] M. Beverland, V. Kliuchnikov, and E. Schoute, "Surface code compilation via edge-disjoint paths," *PRX Quantum*, vol. 3, no. 2, May 2022.

[8] M. E. Beverland, P. Murali, M. Troyer, K. M. Svore, T. Hoefler, V. Kliuchnikov, G. H. Low, M. Soeken, A. Sundaram, and A. Vaschillo, "Assessing requirements to scale to practical quantum advantage," *arXiv preprint arXiv:2211.07629*, 2022.

[9] D. Bluvstein, S. J. Evered, A. A. Geim, S. H. Li, H. Zhou, T. Manovitz, S. Ebadi, M. Cain, M. Kalinowski, D. Hangleiter, J. P. Bonilla Ataides, N. Maskara, I. Cong, X. Gao, P. Sales Rodriguez, T. Karolyshyn, G. Semeghini, M. J. Gullans, M. Greiner, V. Vuletić, and M. D. Lukin, "Logical quantum processor based on reconfigurable atom arrays," *Nature*, vol. 626, no. 7997, pp. 58–65, 2024.

[10] H. Bombin and M. A. Martin-Delgado, "Optimal resources for topological two-dimensional stabilizer codes: Comparative study," *Phys. Rev. A*, vol. 76, p. 012305, Jul 2007. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.76.012305

[11] H. Bombin, I. H. Kim, D. Litinski, N. Nickerson, M. Pant, F. Pastawski, S. Roberts, and T. Rudolph, "Interleaving: Modular architectures for fault-tolerant photonic quantum computing," *arXiv preprint arXiv:2103.08612*, 2021.

[12] J. E. Bourassa, R. N. Alexander, M. Vasmer, A. Patil, I. Tzitrin, T. Matsuura, D. Su, B. Q. Baragiola, S. Guha, G. Dauphinais, K. K. Sabapathy, N. C. Menicucci, and I. Dhand, "Blueprint for a scalable photonic fault-tolerant quantum computer," *Quantum*, vol. 5, p. 392, Feb. 2021. [Online]. Available: https://doi.org/10.22331/q-2021-02-04-392

[13] G. Boyd, "Low-overhead parallelisation of lcu via commuting operators," *arXiv preprint arXiv:2312.00696*, 2023.

[14] S. Bravyi, A. W. Cross, J. M. Gambetta, D. Maslov, P. Rall, and T. J. Yoder, "High-threshold and low-overhead fault-tolerant quantum memory," *Nature*, vol. 627, no. 8005, p. 778–782, Mar. 2024. [Online]. Available: http://dx.doi.org/10.1038/s41586-024-07107-7

[15] S. Bravyi and J. Haah, "Magic-state distillation with low overhead," *Physical Review A*, vol. 86, no. 5, p. 052329, 2012.

[16] S. Bravyi and A. Kitaev, "Universal quantum computation with ideal clifford gates and noisy ancillas," *Phys. Rev. A*, vol. 71, p. 022316, Feb 2005. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.71.022316

[17] S. B. Bravyi and A. Y. Kitaev, "Quantum codes on a lattice with boundary," *arXiv preprint quant-ph/9811052*, 1998.

[18] B. J. Brown, K. Laubscher, M. S. Kesselring, and J. R. Wootton, "Poking holes and cutting corners to achieve Clifford gates with the surface code," *Physical Review X*, vol. 7, no. 2, p. 021029, 2017.

[19] K. R. Brown, J. Kim, and C. Monroe, "Co-designing a scalable quantum computer with trapped atomic ions," *npj Quantum Information*, vol. 2, no. 1, pp. 1–10, 2016.

[20] I. Byun, J. Kim, D. Min, I. Nagaoka, K. Fukumitsu, I. Ishikawa, T. Tanimoto, M. Tanaka, K. Inoue, and J. Kim, "Xqsim: modeling cross-technology control processors for 10+ k qubit quantum computers," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 366–382.

[21] Z. Cai, A. Siegel, and S. Benjamin, "Looped pipelines enabling effective 3d qubit lattices in a strictly 2d device," *PRX Quantum*, vol. 4, no. 2, p. 020345, 2023.

[22] C. Chamberland and E. T. Campbell, "Universal quantum computing with twist-free and temporally encoded lattice surgery," *PRX Quantum*, vol. 3, no. 1, Feb. 2022.

[23] C. Chamberland, K. Noh, P. Arrangoiz-Arriola, E. T. Campbell, C. T. Hann, J. Iverson, H. Putterman, T. C. Bohdanowicz, S. T. Flammia, A. Keller, G. Refael, J. Preskill, L. Jiang, A. H. Safavi-Naeini, O. Painter, and F. G. Brandão, "Building a fault-tolerant quantum computer using concatenated cat codes," *PRX Quantum*, vol. 3, no. 1, Feb. 2022. [Online]. Available: http://dx.doi.org/10.1103/PRXQuantum.3.010329

[24] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, "Topological quantum memory," *Journal of Mathematical Physics*, vol. 43, no. 9, pp. 4452–4505, 2002.

[25] C. Duckering, J. M. Baker, D. I. Schuster, and F. T. Chong, "Virtualized logical qubits: A 2.5 D architecture for error-corrected quantum computing," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 173–185.

[26] A. Erhard, H. Poulsen Nautrup, M. Meth, L. Postler, R. Stricker, M. Stadler, V. Negnevitsky, M. Ringbauer, P. Schindler, H. J. Briegel, R. Blatt, N. Friis, and T. Monz, "Entangling logical qubits with lattice surgery," *Nature*, vol. 589, no. 7841, p. 220–224, Jan. 2021. [Online]. Available: http://dx.doi.org/10.1038/s41586-020-03079-6

[27] A. G. Fowler, S. J. Devitt, and C. Jones, "Surface code implementation of block code state distillation," *Scientific reports*, vol. 3, no. 1, p. 1939, 2013.

[28] A. G. Fowler and C. Gidney, "Low overhead quantum computation using lattice surgery," *arXiv preprint arXiv:1808.06709*, 2018.

[29] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Physical Review A*, vol. 86, no. 3, p. 032324, 2012.

[30] A. G. Fowler, A. C. Whiteside, and L. C. Hollenberg, "Towards practical classical processing for the surface code," *Physical Review Letters*, vol. 108, no. 18, p. 180501, 2012.

[31] X. Fu, L. Riesebos, M. A. Rol, J. van Straten, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, V. Newsum, K. K. L. Loh, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudéver, L. DiCarlo, and K. Bertels, "eQASM: An executable quantum instruction set architecture," in *25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019, Washington, DC, USA, February 16-20, 2019*. IEEE, 2019, pp. 224–237. [Online]. Available: https://doi.org/10.1109/HPCA.2019.00040

[32] C. Gidney, "Tetrationally compact entanglement purification," *arXiv e-prints*, pp. arXiv–2311, 2023.

[33] C. Gidney and M. Ekerå, "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits," *Quantum*, vol. 5, p. 433, 2021.

[34] C. Gidney, N. Shutty, and C. Jones, "Magic state cultivation: growing t states as cheap as cnot gates," *arXiv preprint arXiv:2409.17595*, 2024.

[35] J. J. Goings, A. White, J. Lee, C. S. Tautermann, M. Degroote, C. Gidney, T. Shiozaki, R. Babbush, and N. C. Rubin, "Reliably assessing the electronic structure of cytochrome p450 on today's classical computers and tomorrow's quantum computers," *Proceedings of the National Academy of Sciences*, vol. 119, no. 38, p. e2203533119, 2022. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.2203533119

[36] J. Haah and M. B. Hastings, "Codes and protocols for distilling t, controlled-s, and toffoli gates," *Quantum*, vol. 2, p. 71, 2018.

[37] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," *Physical Review Letters*, vol. 103, no. 15, p. 150502, 2009.

[38] Y. Hirano, Y. Suzuki, and K. Fujii, "Magicpool: Dealing with magic state distillation failures on large-scale fault-tolerant quantum computer," *arXiv preprint arXiv:2407.07394*, 2024.

[39] A. Holmes, Y. Ding, A. Javadi-Abhari, D. Franklin, M. Martonosi, and F. T. Chong, "Resource optimized quantum architectures for surface code implementations of magic-state distillation," *Microprocessors and Microsystems*, vol. 67, p. 56–70, Jun. 2019. [Online]. Available: http://dx.doi.org/10.1016/j.micpro.2019.02.007

[40] A. Holmes, M. R. Jokar, G. Pasandi, Y. Ding, M. Pedram, and F. T. Chong, "NISQ+: Boosting quantum computing power by approximating quantum error correction," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 556–569.

[41] C. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter, "Surface code quantum computing by lattice surgery," *New Journal of Physics*, vol. 14, no. 12, p. 123011, 2012.

[42] A. Kitaev, "Fault-tolerant quantum computation by anyons," *Annals of Physics*, vol. 303, no. 1, pp. 2–30, Jan. 2003.

[43] S. Krinner, N. Lacroix, A. Remm, A. Di Paolo, E. Genois, C. Leroux, C. Hellings, S. Lazar, F. Swiadek, J. Herrmann, G. J. Norris, C. K. Andersen, M. Müller, A. Blais, C. Eichler, and A. Wallraff, "Realizing repeated quantum error correction in a distance-three surface code," *Nature*, vol. 605, no. 7911, pp. 669–674, May 2022. [Online]. Available: http://dx.doi.org/10.1038/s41586-022-04566-8

[44] J. Lee, D. W. Berry, C. Gidney, W. J. Huggins, J. R. McClean, N. Wiebe, and R. Babbush, "Even more efficient quantum computations of chemistry through tensor hypercontraction," *PRX Quantum*, vol. 2, no. 3, Jul. 2021. [Online]. Available: https://doi.org/10.1103/prxquantum.2.030305

[45] A. Li, S. Stein, S. Krishnamoorthy, and J. Ang, "Qasmbench: A low-level quantum benchmark suite for nisq evaluation and simulation," *ACM Transactions on Quantum Computing*, 2022.

[46] L. Lin and Y. Tong, "Heisenberg-limited ground-state energy estimation for early fault-tolerant quantum computers," *PRX Quantum*, vol. 3, no. 1, p. 010318, 2022.

[47] D. Litinski, "A game of surface codes: Large-scale quantum computing with lattice surgery," *Quantum*, vol. 3, p. 128, 2019.

[48] D. Litinski, "Magic state distillation: Not as costly as you think," *Quantum*, vol. 3, p. 205, 2019.

[49] G. H. Low and I. L. Chuang, "Optimal hamiltonian simulation by quantum signal processing," *Physical Review Letters*, vol. 118, no. 1, p. 010501, 2017.

[50] G. H. Low and I. L. Chuang, "Hamiltonian simulation by qubitization," *Quantum*, vol. 3, p. 163, 2019.

[51] J. M. Martyn, Z. M. Rossi, A. K. Tan, and I. L. Chuang, "Grand unification of quantum algorithms," *PRX Quantum*, vol. 2, no. 4, Dec. 2021. [Online]. Available: https://doi.org/10.1103/prxquantum.2.040203

[52] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim, "Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects," *Physical Review A*, vol. 89, no. 2, p. 022317, 2014.

[53] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[54] D. A. Patterson and J. L. Hennessy, *Computer organization and design ARM edition: the hardware software interface*. Morgan kaufmann, 2016.

[55] D. Rosenberg, D. Kim, R. Das, D. Yost, S. Gustavsson, D. Hover, P. Krantz, A. Melville, L. Racz, G. O. Samach, S. J. Weber, F. Yan, J. L. Yoder, A. J. Kerman, and W. D. Oliver, "3D integrated superconducting qubits," *npj quantum information*, vol. 3, no. 1, p. 42, 2017.

[56] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Phys. Rev. A*, vol. 52, pp. R2493–R2496, Oct 1995. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.52.R2493

[57] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Review*, vol. 41, no. 2, pp. 303–332, 1999.

[58] V. V. Sivak, A. Eickbusch, B. Royer, S. Singh, I. Tsioutsios, S. Ganjam, A. Miano, B. L. Brock, A. Z. Ding, L. Frunzio, S. M. Girvin, R. J. Schoelkopf, and M. H. Devoret, "Real-time quantum error correction beyond break-even," *Nature*, vol. 616, no. 7955, pp. 50–55, 2023.

[59] K. N. Smith, G. S. Ravi, J. M. Baker, and F. T. Chong, "Scaling superconducting quantum computers with chiplet architectures," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1092–1109.

[60] A. Steane, "Multiple-particle interference and quantum error correction," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 452, no. 1954, p. 2551–2577, Nov. 1996. [Online]. Available: http://dx.doi.org/10.1098/rspa.1996.0136

[61] S. Stein, S. Sussman, T. Tomesh, C. Guinn, E. Tureci, S. F. Lin, W. Tang, J. Ang, S. Chakram, A. Li, M. Martonosi, F. Chong, A. A. Houck, I. L. Chuang, and M. Demarco, "Hetarch: Heterogeneous microarchitectures for superconducting quantum systems," in *56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23. ACM, Oct. 2023. [Online]. Available: http://dx.doi.org/10.1145/3613424.3614300

[62] Y. Suzuki, T. Sugiyama, T. Arai, W. Liao, K. Inoue, and T. Tanimoto, "Q3de: A fault-tolerant quantum computer architecture for multi-bit burst errors by cosmic rays," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1110–1125.

[63] Y. Ueno, M. Kondo, M. Tanaka, Y. Suzuki, and Y. Tabuchi, "Qecool: On-line quantum error correction with a superconducting decoder for surface code," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, Dec. 2021, p. 451–456. [Online]. Available: http://dx.doi.org/10.1109/DAC18074.2021.9586326

[64] Y. Ueno, M. Kondo, M. Tanaka, Y. Suzuki, and Y. Tabuchi, "QULATIS: A quantum error correction methodology toward lattice surgery," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, Apr. 2022. [Online]. Available: https://doi.org/10.1109/hpca53966.2022.00028

[65] R. Ur Rasool, H. F. Ahmad, W. Rafique, A. Qayyum, J. Qadir, and Z. Anwar, "Quantum computing for healthcare: A review," *Future Internet*, vol. 15, no. 3, p. 94, 2023.

[66] C. Vuillot, L. Lao, B. Criger, C. G. Almudéver, K. Bertels, and B. M. Terhal, "Code deformation and lattice surgery are gauge fixing," *New Journal of Physics*, vol. 21, no. 3, p. 033028, mar 2019. [Online]. Available: https://dx.doi.org/10.1088/1367-2630/ab0199

[67] Q. Xu, J. P. Bonilla Ataides, C. A. Pattison, N. Raveendran, D. Bluvstein, J. Wurtz, B. Vasić, M. D. Lukin, L. Jiang, and H. Zhou, "Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays," *Nature Physics*, vol. 20, no. 7, p. 1084–1090, Apr. 2024. [Online]. Available: http://dx.doi.org/10.1038/s41567-024-02479-z

[68] N. Yoshioka, T. Okubo, Y. Suzuki, Y. Koizumi, and W. Mizukami, "Hunting for quantum-classical crossover in condensed matter problems," *npj Quantum Information*, vol. 10, no. 1, Apr. 2024. [Online]. Available: http://dx.doi.org/10.1038/s41534-024-00839-4

[69] J. Zhang, Z.-Y. Chen, Y.-J. Wang, B.-H. Lu, H.-F. Zhang, J.-N. Li, P. Duan, Y.-C. Wu, and G.-P. Guo, "Demonstrating a universal logical gate set on a superconducting quantum processor," *arXiv preprint arXiv:2405.09035*, 2024.