IRIS: A Compiler for Distributed Quantum Systems

Won Joon Yun
wonjoon.yun@utexas.edu
The University of Texas at Austin

Dhilan Nag
dhilannag@utexas.edu
The University of Texas at Austin

Sneha Ballabh
snehaballabh@utexas.edu
The University of Texas at Austin

Jiapeng Zhao
penzhao2@cisco.com
Cisco Quantum Lab

Eneet Kaur
ekaur@cisco.com
Cisco Quantum Lab

Poulami Das
poulami.das@utexas.edu
The University of Texas at Austin

Abstract

Distributed Quantum Computers (DQCs) enable large system sizes by connecting smaller chips via photonic interconnects. DQCs use teleportation to relocate qubits and execute CNOTs between qubits on different chips. But, non-local CNOTs are 4.3-7.7× slower and 4× more error-prone than local CNOTs within a chip, which degrades program fidelities. Compilers group CNOTs with overlapping qubits into *blocks* and collectively optimize teleportations for each block. However, block-level scheduling suffers from two key drawbacks. *First*, they lack lookahead ability between blocks because they select the best schedule for a block before proceeding. So, they cannot assess the impact of a teleportation on future blocks. Our studies show that naively expanding the lookahead window to include subsequent blocks does not address the issue. *Second*, they do not schedule future block operations or teleportations they need unless preceding blocks are fully scheduled, introducing delay and latency overheads.

We propose *IRIS*, a DQC compiler that addresses these limitations using two key insights: *Utility-driven Lookahead With Multi-Candidate Block Scheduling (UMS)* and *EPR-Capacity-Aware Early Scheduling (EES)*. UMS schedules a block by considering only useful future blocks in its lookahead window. A future block has utility if it shares overlapping qubits with the current block being scheduled. UMS also maintains multiple schedules during compilation, allowing it to defer commitment to globally sub-optimal schedules early. EES enables IRIS to schedule future operations and their relocations early if EPR resources are available for teleportations. Our evaluations show that IRIS reduces teleportations by 34% on average and by up to 65%; and latency by 2× on average and by up to 2.9× compared to the state-of-the-art DQC compiler.

CCS Concepts: • **Computer systems organization** → **Quantum computing**; **Distributed architectures**.

Keywords: Distributed Quantum Computers, Compilation

EuroSys '27, Rabat, Morocco

ACM ISBN 979-8-4007-2971-3/2027/04
https://doi.org/10.1145/3842654.3848552

ACM Reference Format:
Won Joon Yun, Dhilan Nag, Sneha Ballabh, Jiapeng Zhao, Eneet Kaur, and Poulami Das. 2027. IRIS: A Compiler for Distributed Quantum Systems. In *22nd European Conference on Computer Systems (EuroSys '27), April 19–23, 2027, Rabat, Morocco.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3842654.3848552

1 Introduction

Distributed Quantum Computers (DQCs) overcome low yield and fidelity concerns of large monolithic chips by connecting smaller chips via photonic interconnects [1–6]. DQCs offer a pathway to scale system sizes and recent demonstrations [7–11] have led to their inclusion in industrial and academic road-maps [12–15]. Two-qubit CNOT or *entanglement* operations, critical to attain quantum advantage, between qubits on a chip are identical to monolithic systems. In contrast, non-local CNOTs between different chips use *teleportations* to relocate required qubits to the same chip, after which the CNOT is executed locally [16, 17]. Unfortunately, non-local CNOTs exhibit 4× higher error-rates and take 4.3-7.7× longer than local CNOTs [7, 18]. This degrades program fidelity by increasing the vulnerability to errors and decoherence.

DQC compilers improve program fidelity by selecting qubit relocation paths that minimize teleportations and maximize concurrency [19, 20]. The GP compiler selects the shortest qubit relocation paths for each non-local CNOT [21]. AutoComm [22] selects paths collectively for a *block* of non-local CNOTs between two chips, if they have a qubit in common, such that the total number of relocations needed to execute the block is minimized. QuComm [23] extends AutoComm to form even larger blocks by considering non-local CNOTs spread over multiple chips, as long as sufficient EPR capacity is available on a chip to execute them. The *EPR capacity* denotes the number of external qubits a chip can hold at any given time and is mandatory for teleportations. Existing compilers optimize the schedule for each block by selecting relocation paths that maximize the number of local CNOT gates within the block and minimize the number of teleportations required for the non-local gates collectively.

Block-level scheduling, however, has two key drawbacks. *First*, it lacks lookahead ability between blocks. As teleportations relocate qubits, any teleportation impacts the scheduling of all future gates and teleportations both. Current compilers only minimize teleportations for each block but

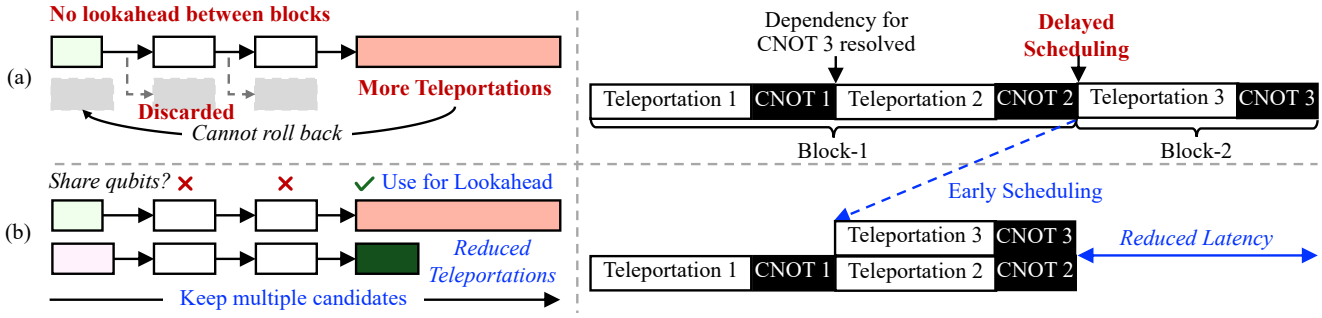

Figure 1. (a) Current compilers have limited lookahead capability between blocks and commit to the schedule with fewest teleportations for each block while discarding the rest before proceeding to the next block. They also defer scheduling future gates until preceding blocks complete. (b) IRIS considers useful future blocks in its lookahead window and retains multiple candidate schedules. It also executes future operations and their teleportation early if sufficient EPR capacity is available.

do not assess how these relocations impact future blocks. So, for a program with N blocks, B_1 to B_N, they cannot evaluate how qubit displacements in B_1 impact teleportations in B_2 to B_N. *Ideally*, a compiler must use all future blocks in the *lookahead window* and assess this impact, but it is not feasible due to exponential complexity. In practice, it may only be able to consider F future blocks ($F \ll N$). Also, unlike monolithic compilers where only considering the next F gates suffices for lookahead heuristics, our studies show that simply using the next F blocks does not work for DQCs. To show this, we consider a program with 20 random blocks. The small size allows us to derive the optimal schedule. By using a lookahead window with the next 10 blocks while scheduling each block, QuComm reduces teleportations from 21 to 16. However, it is still higher than the optimal (13). This is because a chip typically holds many qubits which means several blocks comprise only local CNOTs; and consecutive teleportations on a qubit are often several blocks apart (distance>F). Our studies using a QAOA program with 57K CNOTs grouped into 17K blocks show that there are ~260 CNOTs on average between consecutive relocations on a qubit. With an average block size of ~3 CNOTs here, these two relocations are ~89 blocks apart. Thus, a lookahead window with only a few future blocks does not suffice. Committing to a block schedule early also limits lookahead performance because a schedule that is optimal for a block may increase teleportations in future, as shown in Figure 1(a). Alternate schedules that may reduce teleportations in future are discarded by the compiler.

The second limitation of block-level scheduling is that a block is scheduled only after *all* preceding blocks have been scheduled. Also, teleportations are only scheduled when the compiler encounters a non-local gate (*on-demand scheduling*). This causes future gates and any teleportations they require to wait to be scheduled even if data dependencies have been resolved earlier, increasing latency. For example, if block B_1 unlocks a data dependency for a gate in block B_2, the gate and its teleportations are not scheduled until the compiler has scheduled all operations up to that gate in B_2, as shown in Figure 1(a). Our studies show that 45.4% of teleportations wait for 7.5 ms on average in the same QAOA program above.

We propose *IRIS*, a compiler that addresses these issues by using two key insights. *First*, we design *Utility-Driven Lookahead With Multi-Candidate Block Scheduling (UMS)*. To optimize the lookahead heuristic, UMS leverages a utility-driven approach instead of naively considering N subsequent blocks. A future block has *utility* in the lookahead heuristic of the current block being scheduled, only if both have overlapping qubits. This enables IRIS to optimize teleportation routes with a small lookahead window without drastically increasing compilation complexity. However, evaluating the impact of a current block schedule on the teleportation overheads in future blocks is non-trivial because the overhead of an i^{th} future block can only be determined by actually scheduling it starting with qubit positions at the end of the $(i-1)^{\text{th}}$ block. Moreover, once a schedule for the current block is already committed to the final executable, the compiler cannot evaluate alternatives that may have led to fewer teleportations in the future. To address this issue, UMS retains multiple schedules for the current block. Then, starting with each schedule and corresponding qubit positions, IRIS schedules subsequent blocks, eventually building a tree of solutions. For each non-local CNOT, UMS selects teleportation routes with minimal relocations. To keep the complexity tractable, IRIS prunes the tree and retains only Top-w solutions, where w is the maximum number of solutions considered.

Second, we propose *EPR-capacity-aware Early Scheduling (EES)* that schedules future CNOTs and any teleportations they require, if EPR capacity is available, as early as possible. Early scheduling is non-trivial because dependent operations are often many blocks apart and form long chains. Scheduling teleportations early alters EPR resource distributions and may increase teleportation costs of intermediate operations. For example, if a gate in block B_1 resolves the dependency for a CNOT in B_5, scheduling the B_5 gate early may change EPR capacities, increasing relocation costs for operations pending

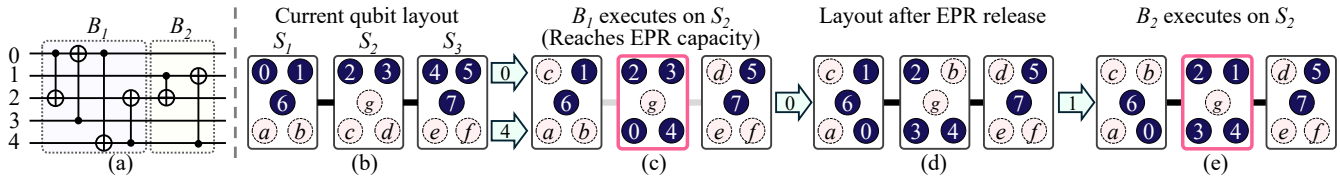

Figure 2. Compilation of (a) a circuit on (b) a 3-chip DQC using state-of-the-art DQC compiler QuComm [23]. QuComm groups the CNOTs into two blocks, B_1 and B_2. (c) It selects S_2 and relocates qubits 0 and 4 in parallel to execute B_1 on S_2, exhausting its EPR resources. (d) To free EPRs, QuComm relocates qubit 0 from S_2 to S_1. (e) It then relocates qubit 1 to S_2 for B_2.

in B_1 to B_4. To address this, EES schedules a program, collects the set of gates E that can be scheduled early by tracing instruction dependency chains, and tracks EPR capacities and qubit positions post each operation. Next, EES schedules each operation in E earlier than its current timeline if it does not increase the number of teleportations required. EES ensures that such early scheduling teleportations does not fully consume the EPR capacity of a chip; else, some EPR resources would need to be freed for downstream operations. Figure 1(b) illustrates how IRIS yields fewer teleportations overall by employing utility-driven lookahead and retaining multiple candidate schedules. It also reduces latency by enabling EPR-capacity-aware early scheduling.

Our evaluations using representative benchmarks and DQCs show that IRIS reduces teleportations and latencies by 34% and 2× on average. IRIS is available at: IRIS-Code.

To that end, this paper makes the following contributions:

1. We design *IRIS*, a compiler to reduce the overheads of teleportations in Distributed Quantum Computers (DQCs).
2. We propose *Utility-Driven Lookahead with Multi-Candidate Block Scheduling (UMS)*, which identifies the relevant future blocks that have utility in the lookahead heuristics of each block and manages multiple schedules at any given time to prevent convergence on sub-optimal schedules.
3. We propose *EPR-capacity-aware Early Scheduling (EES)* that schedules future operations and any relocations required by them as soon as their data dependencies are resolved.

2 Background

2.1 Distributed Quantum Computers (DQCs)

DQCs connect smaller quantum chips via photonic channels. Typically, chips have nearest-neighbor connectivity because all-to-all connectivity is expensive and reduces fidelity [5–7, 24–26]. Non-local CNOTs between qubits on different chips use teleportations [16, 17]. Teleportations require EPR pairs, for which qubits must be reserved. We provide more details on EPR pair generation in Appendix A. Figure 2(b) shows a DQC with five qubits per chip, blue qubits are *compute* qubits (0–7), while the others (a–g) are communication qubits used to hold external qubit's quantum state or remotely control a qubit. We consider two types of teleportations:

RELOCATE [16]: or *state teleportation* transfers the state of a qubit from one chip to another. For a non-local CNOT between adjacent chips, a RELOCATE moves one of the qubits to the other chip, after which the CNOT becomes local. For non-adjacent chips, multiple RELOCATEs are used to bring both qubits on the same chip. RELOCATEs displace qubits and the number of external qubits a chip can hold equals the number of communication qubits or *EPR capacity*.

Remote CNOT (Re-CNOT) [17]: or *gate teleportation* enables non-local CNOTs between adjacent chips without relocating a qubit. For example, to perform a CNOT between qubits 0 and 2 in Figure 2(b), we may RELOCATE 0 to S_2 using EPR pair b-c. Qubit c holds the state of 0 after the RELOCATE. If we were to use a Re-CNOT instead, qubit 0 remains on S_1. The EPR pair is still required for the Re-CNOT as one of its qubits serves as the proxy control for the CNOT. For a CNOT between qubits 0 and 4, we must either relocate both qubits to S_2 or relocate 0 to S_3 via S_2 because S_1 and S_3 are not connected, thus requiring multiple RELOCATEs.

We provide details of these operations in Appendix B.

2.2 The Problem: Teleportation Overheads

RELOCATEs and Re-CNOTs take 4.3× to 7.7× longer and have 4× higher error rates than local CNOTs [7, 18]. Although, programs often use far fewer teleportations than local CNOTs, the former dominates latency. Our studies with a 120-qubit QAOA program on a 2×2 DQC show that even though only 6% of operations are teleportations, they consume 50% of the latency because teleportations have much lower concurrency (1.1) than local CNOTs (3.9) because teleportations are often serialized due to the limited connectivity between chips on DQCs and availability of EPR resources.

2.3 Prior Works on DQC Compilation

Compilers minimize teleportations and maximize concurrency. Teleportations displace qubits, and each displacement can make subsequent CNOTs local or non-local. The GP compiler schedules each non-local CNOT independently along the shortest relocation path [21]. This minimizes teleportation for the CNOT but ignores how the displacement affects subsequent CNOTs. AutoComm addresses this by scheduling multiple CNOTs together, so that one displacement can benefit them all. AutoComm groups non-local CNOTs that share a qubit and span two chips into a *block* [22], and schedules

the block as a whole. For example, CNOT 0,2 and CNOT 3,0 in Figure 2(b) share qubit 0. Scheduling them individually could cost up to two RELOCATEs, while scheduling them as a block costs one, because relocating qubit 0 to S_2 makes both CNOTs local. QuComm [23] extends AutoComm further. It forms larger blocks spanning multiple chips, as long as the block does not exceed the EPR capacity of a chip. For each block, QuComm constructs a *block schedule*, a sequence of teleportations and local CNOTs that executes all gates in the block. QuComm generates this block schedule in two steps:

Block Formation: QuComm groups non-CNOTs into blocks, such that the EPR capacity required to execute each block is available on a single chip. It also merges local CNOTs into the blocks so that RELOCATEs do not make them non-local.

Block Scheduling: For each non-local CNOT, QuComm decides whether to use RELOCATE or Re-CNOT. If RELOCATEs are used, it identifies the qubits to relocate and the chip to relocate to, such that the total number of RELOCATEs required for the current gate and the remaining gates in the block are minimized. If a target chip has only one communication qubit (such as communication qubit g on chip S_2 in Figure 2(c)), QuComm evicts some of its current qubits to other chips using an operation called *EPR release*. For this, it prioritizes qubits whose relocations assist future blocks.

We explain QuComm using Figure 2. QuComm groups the CNOTs into blocks B_1 and B_2. For B_1, QuComm selects chip S_2 and relocates qubits 0 and 4 in parallel, requiring fewest RELOCATEs and yielding maximum concurrency. Scheduling on S_1 costs four RELOCATEs (2, 3, and 4), two of which are serialized (relocating 4 to S_1 via S_2), whereas scheduling on S_3 requires three RELOCATEs (0, 2), two of which are serialized (relocating 0 to S_3 via S_2). Block B_2 cannot be scheduled because EPR resources on S_2 are exhausted. QuComm relocates qubit 0 to S_1 to release EPR capacity, after which qubit 1 relocates to S_2, where the block executes.

3 Limitations of Prior Works and Insights

Block-level scheduling in prior works suffers from two issues.

3.1 No Lookahead Between Blocks

A teleportation impacts relocation costs of all future teleportations, whether they belong to non-local gates within the same block that is being scheduled or future blocks. For example, if qubit q is relocated in block B_0, it impacts the scheduling of all remaining gates in B_0. Moreover, if the same qubit must be relocated again while scheduling block B_{10}, then the relocation costs of B_{10} depend on the position of q based on teleportations used in B_0 scheduling. Thus, while selecting teleportations for q during B_0 scheduling, we must satisfy two objectives: (1) minimize relocation costs of the current block B_0 (***Objective-1***) and (2) minimize relocation costs of future blocks (such as B_{10}) (***Objective-2***). An ***ideal lookahead*** DQC compiler must optimize for both objectives for all teleportations. But, this is impractical as its complexity scales exponentially with the size of programs. Existing compilers, like QuComm, employ *low-cost lookahead heuristics* that only optimize for Objective-1 to schedule a block. They commit this schedule to the executable and only then proceed to the next block. Thus, they lack *lookahead ability between blocks* and only optimize for Objective-2.

Lookahead heuristics, well studied for monolithic systems, use a set of *subsequent* gates in the lookahead window while scheduling a gate [27]. But naively adopting similar approaches, where the lookahead window comprises *the next F future blocks* is insufficient in DQCs for two reasons. *First*, programs comprise several blocks with only local gates in between two consecutive teleportations on the same qubit. Including these *local-only* blocks in the lookahead window offer no practical benefits. *Second*, consecutive teleportations on a qubit are often many blocks apart. So, the impact of a RELOCATE on future teleportations is only felt several blocks ahead. Small lookahead windows, essential for low compilation complexity, miss these *far-in-the-future* blocks.

Table 1 shows the average CNOT, block, and local-only block counts in between two consecutive RELOCATEs on a qubit in a 3×3 DQC. Consider the QAOA-FC benchmark with 57K CNOTs grouped into 17K blocks. On average, two consecutive RELOCATEs are separated by ∼90 blocks comprising ∼260 CNOTs, out of which ∼72 blocks have only local gates. Across all programs, consecutive RELOCATEs on a qubit are separated by 55-112 blocks on average. Small lookahead windows with only the next 10-15 blocks are thus, inadequate. Now one could argue that the overlapping qubit could be evicted from its location by an EPR release while scheduling the 17 (=89-72) blocks with non-local gates in between, rendering long-range lookahead ineffective. However, note that the likelihood of this is extremely low because only <1 EPR release occurs on average between consecutive RELOCATEs and the likelihood of using the overlapping qubit is extremely low (given each chip has hundreds of qubits).

Table 1. Average number of CNOTs, blocks, local-only blocks, and EPR releases between two consecutive RELOCATEs on a qubit in a 3×3 DQC for 240-qubit programs.

Program	CNOTs	Blocks	Local-Only	EPR releases
QAOA-FC	261.6	89.6	72.1	0.12
QFT	137.2	55.3	33.2	0.02
QV	240.6	112.3	64.8	0.67
Shor	233.3	77.5	58.4	0.03
VQE	277.7	112.0	91.3	0.06

The early commitment to a schedule also acts as a critical barrier because a schedule that is optimal for a block may eventually become *globally sub-optimal* as future blocks are scheduled, as shown in Figure 4. Unfortunately, existing compilers have no mechanisms to detect this sub-optimality and

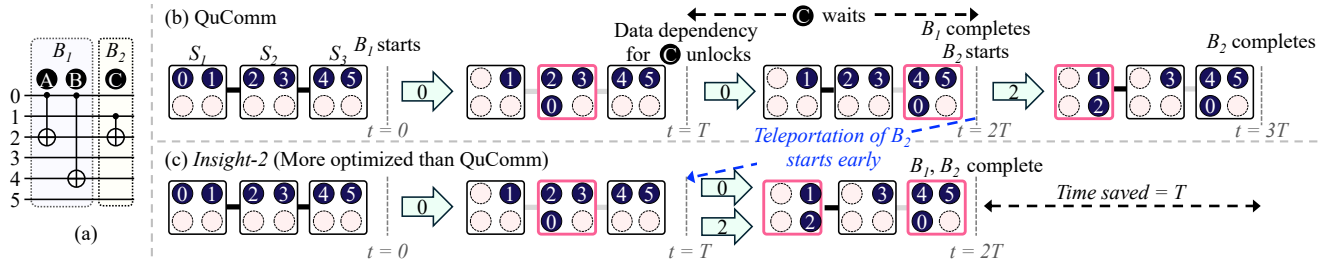

Figure 3. (a) An example circuit with blocks B_1 and B_2. (b) Dependency for CNOT C is resolved after executing CNOT A. But QuComm does not schedule C until it reaches this gate in B_2 for scheduling. The teleportation required is also scheduled only when the compiler realizes that a relocation is necessary for scheduling C. (c) In contrast, IRIS relocates qubit 2 and schedules CNOT C *as early as possible*. This improves RELOCATE and CNOT concurrency and reduces overall program execution time.

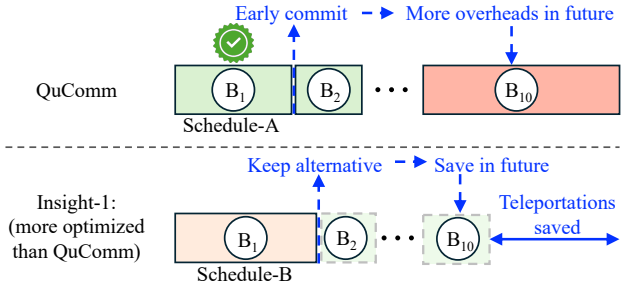

Figure 4. QuComm selects Schedule-A for B_1 as it costs fewer teleportations (illustrated by the box width here) than Schedule-B. But qubit relocations in Schedule-A eventually cost more teleportations in future block B_{10}. In contrast, IRIS also retains Schedule-B and schedules subsequent blocks, eventually arriving at a more optimized schedule for block B_{10}. This allows IRIS to select block schedules that are locally sub-optimal but globally better (like Schedule-B).

cannot pick an alternative schedule for a block scheduled in the past. For example, if schedule A requires fewer teleportations than schedule B for block B_1, QuComm selects A before proceeding forward. However, it is possible that schedule B would have resulted in significantly fewer teleportations for block B_{10} and minimized teleportations overall. QuComm has no mechanisms to evaluate this because it discards B.

Insight-1: Practical utility-driven lookahead tailored to DQCs: DQC compilers must consider *utility-driven lookahead*, where the lookahead window comprises only *useful* future blocks rather than the next F blocks. Moreover, to evaluate the cost of a relocation far ahead in future, compilers must avoid early commitment to a block schedule, and work with multiple schedules in parallel. Lastly, compilers must enable these optimizations without losing tractability.

3.2 Delayed Operation Scheduling

Blocks comprise many gates that execute at different times, with some finishing sooner than others. The early operations of a block often resolve data dependencies for operations in future blocks. However, current compilers defer scheduling these gates until the compiler arrives at the corresponding future blocks for scheduling. Compilers also schedule teleportations *on-demand*, which means that they determine the relocation paths only when they schedule a non-local CNOT and teleportations are required. These factors severely delay instruction scheduling, particularly long-latency teleportations, increasing program latencies. Our experiments across five benchmarks show that 51% of teleportations are delayed on average, and each delayed teleportation waits for about 18 ms on a 3×3 DQC architecture, as shown in Table 2.

Table 2. Percentage of delayed teleportations and their average wait time on a 3 × 3 DQC with 240-qubit programs.

Metric	QAOA-FC	QFT	QV	Shor	VQE
Delayed teleportations (%)	45.4	55.1	51.9	53.0	50.3
Avg. wait time (ms)	7.5	24.6	11.9	36.3	11.1

We explain this with Figure 3. Even though the data dependency for CNOT C is resolved when CNOT A executes (t=T), existing compilers wait to schedule it until block B_1 completes (t=$2T$), as shown in Figure 3(b). Also, the RELOCATE for CNOT C is only scheduled when the compiler arrives at that CNOT in B_2. In contrast, scheduling the RELOCATE and CNOT earlier, as shown in Figure 3(c), reduces overall latency. The benefits compound in larger programs comprising hundreds to thousands of blocks because such early scheduling eventually leads to a cascade of operations to be scheduled sooner. For example, scheduling CNOT C early unlocks data dependencies in subsequent blocks.

Insight-2: Decouple teleportations from CNOTs and schedule across block boundaries: DQC compilers must decouple teleportation scheduling from CNOT scheduling so that qubits are relocated ahead of time whenever feasible and EPR capacity is available. This allows the qubits to be available locally when the compiler is actually ready to schedule the corresponding CNOT. Compilers must also schedule gates as soon as possible, breaking the abstraction of the block boundary. This improves the instruction-level concurrency and reduces program runtimes.

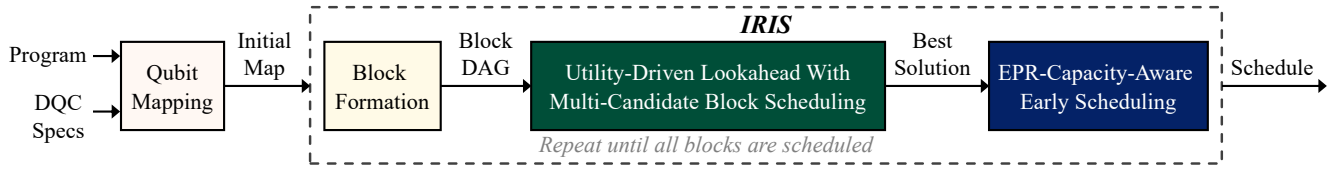

Figure 5. Overview of IRIS. It takes as input the program and initial qubit mapping and mainly focuses on instruction scheduling. This involves block formation (grouping operations into instruction blocks), an utility-driven lookahead-based scheduling involving multiple candidates, and optimizing teleportations via EPR-capacity-aware early scheduling.

4 Our Proposal: IRIS

We propose *IRIS* that reduces teleportation overheads during program execution by employing the insights described in the previous section. A DQC compiler involves two steps: mapping and scheduling. During mapping, it assigns a physical qubit to each program qubit. During scheduling, it starts with this initial layout to schedule instructions and updates the mapping based on qubit relocations during teleportations. *IRIS mainly focuses on scheduling and its mapping is based on prior works.* Figure 5 gives an overview of IRIS.

Pre-Processing: Before compilation, physical qubits are split into compute (for program qubits) and communication qubits (for EPR pairs). By default, IRIS reserves 90% of chip qubits for compute and 10% for communication as non-local CNOTs occupy 3-8% of operations in most programs.

4.1 Qubit Mapping

IRIS maps qubits in three steps: (1) partitions a program graph into N sub-graphs (for DQC with N chips), (2) assigns each sub-graph to a chip, and (3) maps each program qubit of a sub-graph to a compute qubit of the selected chip.

Step 1– Program partitioning: This step minimizes non-local CNOTs based on the initial qubit layout. Partitioning methods can be broadly classified into two types: *static* and *dynamic*. We consider both because IRIS is orthogonal to mapping and focuses on scheduling. Static methods, such as OEE [28] and Min-Cut [29], partition the program at the beginning of compilation and fix the initial layout prior to scheduling. However, as scheduling proceeds and qubits relocate, some non-local CNOTs become local. Static methods cannot optimize RELOCATEs further based on dynamic qubit layout transitions as scheduling proceeds. So, IRIS also features state-of-the-art dynamic partitioning methods, such as WBCP [30] and GCP [31]. They address this issue by dividing a program into multiple segments and selecting a separate qubit layout for each segment. As consecutive segment layouts may differ, these methods optimize for both the number of non-local CNOTs within each segment and RELOCATEs needed to transform one segment layout to the next.

By default, IRIS uses Min-Cut partitioning [29]. While dynamic methods outperform static methods individually in general, our evaluations show that Min-Cut outperforms other mappers because the scheduler in IRIS outperforms the built-in scheduling integral to the dynamic schemes. We discuss these details in our evaluations.

Step 2– Subgraph-to-chip mapping: Typically, DQCs have nearest-neighbor connectivity and sub-graphs with many non-local CNOTs between them must be placed on physically nearby chips to reduce collective RELOCATE costs [30]. IRIS adopts the Integer Linear Programming (ILP) formulation proposed by Kaur et al [30] because it is a state-of-the-art.

Step 3– Subgraph-to-compute qubit mapping: This step assigns program qubits to compute qubits on a chip. It only affects the cost of on-chip gates depending on the qubit technology, and does not impact teleportation costs. IRIS applies a trivial mapping from program qubits within each chip to compute qubits and leaves optimization to each platform's monolithic compiler and is consistent with prior works [22, 23].

4.2 Block Formation

IRIS initiates block formation by converting a program into a Directed Acyclic Graph (DAG) that captures data dependencies. The Block Formation Algorithm (BFA) starts with a current block C set to the first gate g_0. BFA aggregates subsequent gates from the DAG that share one or more qubits with gates in C (called overlapping qubits) into a candidate block D. BFA fuses C and D if the merged block, $C'=[C+D]$, can be executed on a chip with lower teleportation costs than executing C and D independently, and then updates C to C'. If they cannot be merged due to insufficient EPR resources, BFA removes the last gate from D and retries merging C and D. If D is empty or the EPR capacity of the chip where C is expected to execute has been reached, BFA saves C as block B_0, sets C to the next unassigned gate, and continues until all gates are assigned to blocks (B_is).

The *cost* (c) of executing a block on a chip is computed as the number of RELOCATEs (n_{relocate}) and Re-CNOTs (n_{remote}), using Equation (1). Re-CNOTs take longer than RELOCATEs. We account for this using the parameter α. By default, we set α=1.77 based on their latencies on neutral atom systems [7, 18]. If there are multiple relocation paths to execute a block on a chip, BFA selects the one with minimum cost.

$$c = n_{\text{relocate}} + \alpha \times n_{\text{remote}}, \tag{1}$$

We explain BFA using Figure 6. BFA starts with C=[g_0] and D=[g_1] because g_1 has qubit 2 overlapping with C. The

cost of executing C on either chip is 1 RELOCATE (qubit 0 from A to B or qubit 2 from B to A). Similarly, the cost of executing D is 1 RELOCATE. Executing the fused block $C'=[g_0, g_1]$ costs 1 RELOCATE. Thus, C is updated to $[g_0, g_1]$ and D is set to $[g_2]$ (qubit 1 is the overlapping qubit). But the cost of the combined block $C' = [g_0, g_1, g_2]$, is now 2 RELOCATEs, exceeding EPR capacity. So, the fusion stops.

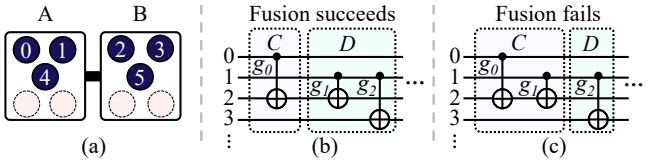

Figure 6. (a) A 1 × 2 DQC. (b) BFA fuses blocks C and D because the cost of executing the combined block is lower than executing them separately and the required EPR capacity is available. (c) BFA does not fuse C and D because the execution cost of combined block exceeds the EPR capacity.

4.3 Utility-Driven Lookahead With Multi-Candidate Block Scheduling

Next, starting with the initial qubit layout, IRIS proceeds to schedule each block. We propose *Utility-Driven Lookahead With Multi-Candidate Block Scheduling (UMS)*. It employs two features: a *utility-driven* lookahead window and the ability to retain multiple schedules in a solution tree.

4.3.1 Utility-Driven Lookahead Window Selection.

The *utility* of a *future* block (F) determines the benefit of including it in the lookahead window of the current block (C), computed using Equation (2). If both blocks have overlapping qubit(s) and the compiler relocates at least one of them while scheduling C, there is a likelihood that the updated qubit positions will impact the relocations needed by F.

$$\text{Utility}(F) = 1;\ \text{if } Q \neq \emptyset,\ \text{where } Q = \{q \mid q \in F \cap C\} \quad (2)$$

IRIS sets the next unscheduled block as the current block C and prepares its lookahead window. It parses each subsequent block and checks for overlapping qubits. If there exists at least one overlapping qubit, the future block has *utility* for C's lookahead heuristic and is added to its lookahead window. The process repeats until no blocks are left in the program or the lookahead window size is reached. Figure 7 illustrates this. As scheduling begins, C is set to the first block B_1. The lookahead window does not include blocks B_2 and B_3 but includes B_4 because it has overlapping qubit (q_0) with C. The current block and its lookahead window form the scheduling group G (i.e. G= $[C, B_4]$). By default, IRIS uses a lookahead window with 4 blocks, allowing it employ lookahead heuristics while keeping the complexity tractable (illustration shows only one block for brevity).

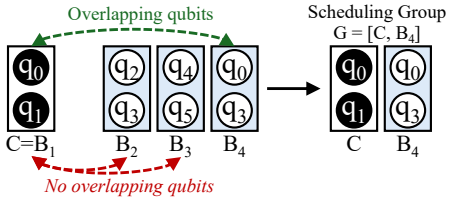

Figure 7. Illustration of utility-driven lookahead. Block B_4 has an overlapping qubit with current block C and has utility in its lookahead but B_2 and B_3 do not. Scheduling group G is the set of the current block and its lookahead window.

4.3.2 Multi-Candidate Block Scheduling.

IRIS starts with an initially empty solution tree and maintain at most w schedules at any time, where each branch or path corresponds to a schedule. As scheduling proceeds, a layer of nodes is added for every non-local CNOT. Each node comprises the schedule between two consecutive non-local CNOTs, qubit layout at the end of the corresponding schedule, and EPR capacity distributions. IRIS schedules the current block C in instruction order. Let g be the next unscheduled gate. If g is a local CNOT, it is scheduled and added to the newest layer of nodes. If g is a non-local CNOT, UMS evaluates teleportation routes by considering (a) which chips to execute g and (b) which qubits to relocate. For each combination of these, UMS selects between RELOCATEs and Re-CNOTs depending on what is cheaper, identifies relocation paths, and checks if EPR capacity is available. If the EPR capacity of a chip along the relocation path is exhausted, UMS frees EPR capacity using the *EPR release* and prioritizes the eviction of qubits whose relocation benefits future gates in group G. If no such qubit is found, UMS evicts an external qubit. UMS minimizes the cost of scheduling g using Equation (3), computed as the sum of its teleportation costs (C_g), any EPR releases (C_{EPR}), and its impact (C_R) on the set of remaining non-local CNOTs (R) in the scheduling group G.

$$C_{\text{total}}(g) = C_g + C_{\text{EPR}} + C_R. \quad (3)$$

Evaluating C_R is non-trivial because teleportation costs of a future gate cannot be determined without actually scheduling it starting with qubit positions at the end of its preceding gate. So, UMS only *estimates* C_R based on current qubit positions, using Equation (4), where C_i is the teleportation cost of the i^{th} non-local gate in R, β is a decaying factor based on distance (d) between the blocks comprising gates g and i which allows UMS to prioritize future gates based on their distance from current gate. As discussed earlier, EPR releases do not occur frequently and so, the qubit being relocated for g typically stays at the same location when gates in R are scheduled. As a result, our estimated C_R closely matches the actual C_R. By default, we set $\beta = 0.871$ so that a block that is 10 blocks ahead in the future contributes a weight 0.25.

$$C_R = \textstyle\sum_{i \in R} \beta^{d(i)} \times C_i. \quad (4)$$

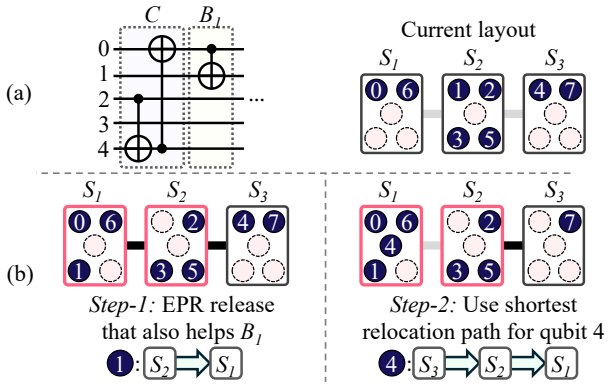

Figure 8. (a) A scheduling group comprising the current block C and its utility-driven lookahead window B_1 being scheduled assuming a current qubit layout. (b) UMS frees EPR resources on chip S_2 by relocating qubit 1 to S_1. This relocation also helps in reducing the teleportation costs of the CNOT in block B_1. To relocate qubit 4, UMS uses the shortest relocation path, allowing it to minimize teleportation costs.

We explain scheduling using Figure 8. Assume a chip has an initial EPR capacity of two and qubits (0, 1), (2, 3), and (4, 5) are mapped to chips S_1, S_2, and S_3, respectively. Although C and B_1 share an overlapping qubit, block formation does not merge them to avoid exceeding the EPR capacity. Consider the scheduling group G=[C, B_1] with C as the current block and the qubit layout in Figure 8. To schedule C, qubit 4 must relocate from S_3 which requires an EPR release on S_2. For this, UMS evicts qubit 1 to S_1 that makes the CNOT in B_1 local. UMS executes the first CNOT in C between qubits 2 and 4 on S_2 along the relocation path of qubit 4.

For each non-local gate g, UMS starts with qubit layout in each node in the last layer (say L_i) of the tree and considers p relocation paths for g, where p is the number of chips comprising qubits of remaining operations in G. For example, if the qubits involved in the operations remaining in G after g are on two chips, UMS considers p=2 relocation paths for g. Once the paths are determined, nodes are added to the next layer (L_{i+1}) of the tree. When UMS completes evaluation for all nodes in layer L_i, the tree is pruned to retain only the top-w solutions. Figure 9 illustrates this process. When gate g_1 is scheduled, UMS considers two relocation paths, adding two nodes to the tree. The next non-local gate g_2 schedules starting from each L_1 node and considers two relocation paths per node. Scheduling proceeds to gate g_3 starting with the L_2 nodes. Assuming each node adds two relocation paths for g_3, the tree now comprises eight paths which exceeds the maximum capacity (w=4) and the tree is pruned to only retain the top four candidates. Once all gates in the current block C are scheduled, IRIS sets C to the next unscheduled block. The process continues until all the blocks in the program are scheduled. After this, IRIS returns the schedule from the tree with the fewest teleportations.

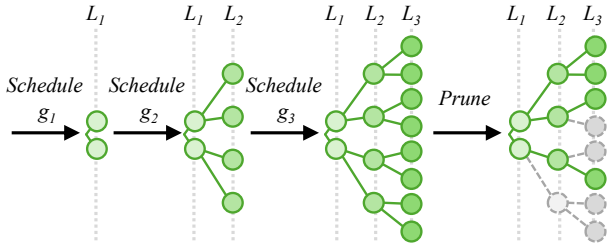

Figure 9. Nodes are added to the solution tree for each non-local gate scheduled depending on relocation paths (here, each non-local CNOT adds two paths). Once the tree exceeds w solutions, it is pruned to only retain the top-w candidates.

4.4 EPR-Capacity-Aware Early Scheduling (EES)

IRIS executes future gates and teleportations early *without waiting* for the current block to complete. Our proposed method, *EPR-Capacity-Aware Early Scheduling (EES)* enables two capabilities: (1) early scheduling of a RELOCATE required for a future CNOT, if the qubit involved in the RELOCATE does not participate in any other operations, even if the data dependencies for the CNOT are not yet resolved, which allows us to decouple teleportation from CNOT scheduling and (2) early scheduling of future CNOTs and any RELOCATEs they need if data dependencies are resolved.

But such early execution is non-trivial because dependent operations are often blocks apart and executing RELOCATEs early may alter EPR capacities and qubit positions that interfere with scheduling intermediate operations. For example, if a gate in block B_1 resolves the dependency for a gate in B_5, scheduling the B_5 gate early may alter qubit positions and EPR resource distributions such that scheduling pending operations in B_1 to B_4 costs extra relocations. These costs are unknown until the gates are scheduled. Also, tracking dependencies is hard as they span several blocks (say $B_1 \rightarrow B_5 \rightarrow B_{10}$). EES overcomes these challenges using a two-pass approach.

First, EES schedules the program using UMS, finds a set of all instructions E that may be executed earlier than their current timelines by tracing their dependency chains, and tracks qubit positions and EPR capacities after each instruction. Next, for each instruction e in E, EES checks its current execution timestamp (T_f) and the earliest timestamp (T_e) it can execute. Then, EES checks if e can be executed a cycle earlier than T_f (at T_i=$T_f - \tau$) without extra relocations. Here, τ is the execution time of the previous instruction. If e needs extra relocations to execute at T_i, EES removes it from E and schedules it at its current timeline. Otherwise, it updates the schedule by shifting e earlier to T_i. If e is a RELOCATE, EES executes it at T_i only if it does not consume the full EPR capacity of the corresponding chip. This ensures subsequent instructions do not need EPR releases. The process continues until all timestamps up to T_e are evaluated. The steps are repeated for all instructions in E before moving forward. After EES processes all instructions, it returns the final schedule.

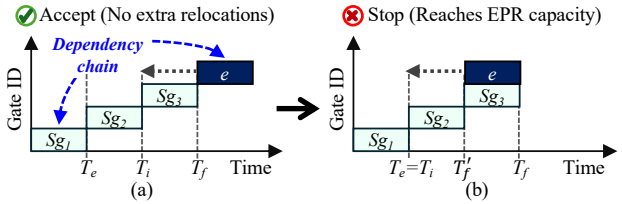

Figure 10. (a) EES finds the earliest timestamp T_e at which a future instruction e can execute. It shifts e from its timestamp T_f to an earlier one, T_i, if it does not add relocations. (b) EES stops shifting e earlier than T_i if a chip reaches EPR capacity.

Figure 10 illustrates EES. Here, e is an early executable instruction whose dependency is resolved after Sg_0 executes. EES shifts the execution of e from T_f to T_i as it does not incur extra relocations. But, EES does not shift the execution to T_e as the EPR capacity of the corresponding chip is reached.

We summarize the IRIS algorithm in Appendix C.

5 Evaluation Methodology

This section describes our evaluation methodology.

5.1 Benchmarks

We evaluate IRIS using a wide array of benchmarks, ranging from arithmetic circuits to system-level benchmarking programs and variational quantum algorithms [32–37]. Program sizes (up to 86.4K CNOTs) vary by DQC size, and far exceed those used in prior works. Table 3 summarizes them.

Table 3. CNOT counts in benchmarks for 2×2, 2×3, and 3×3 DQCs (with 120, 180, and 240 program qubits, respectively).

DQC	BV	QAOA 3reg	QuGAN	VQE	QFT	Shor	QAOA FC	QV
2 × 2	119	360	706	7.2K	7.3K	13.6K	14.3K	21.6K
2 × 3	179	540	1.1K	16.2K	16.4K	31.1K	32.2K	48.6K
3 × 3	239	720	1.4K	28.8K	29.0K	55.9K	57.4K	86.4K

5.2 Baseline

Mapping: We employ WBCP [30], GCP-E [31], static OEE [28] and Min-Cut [29] for circuit partitioning and Kaur's ILP method [30] for mapping sub-graphs onto chips, which represents the state-of-the-art, enabling a competitive baseline.

Program Scheduling: We compare against QuComm [23], the state-of-the-art scheduler for DQCs. As its software is not open sourced, we implement and reproduce it faithfully.

5.3 Hardware Architectures

Architectures: We consider 2×2, 2×3, and 3×3 DQCs to match industry roadmaps [12, 13, 38]. We consider 100–500 qubits systems and varying chip sizes, which far exceed the 300-qubits maximum used in prior works [18, 22, 23, 39, 40].

Error Model: We assume neutral-atoms [18] because they have high photonic link rates and fidelity [7]. We consider dual-atom architectures where Rubidium (^{87}Rb) and Ytterbium (^{171}Yb) atoms are used for compute and communication qubits respectively. This heterogeneity allows frequent measurements and teleportation-related operations on communication qubits while minimizing crosstalk with compute qubits [41–43]. We summarize the parameters in Table 4.

Table 4. Operational latencies in neutral atom DQCs [7, 18]

Chip Params.	Time	Interconnect Params.	Time
1Q gate	52 µs	Measurement	1 ms
2Q gate	360 ns	EPR Pair Generation	259 µs
Atom transfer	15 µs	RELOCATE	1.3 ms
Atom movement	300 µs	Re-CNOT	2.3 ms

Assumptions on EPR Pair Generation: EPR pair generation is probabilistic and error-prone. Producing high-fidelity EPR pairs requires *distillation* performed by repeating a Bell-State Measurement (BSM) process [7, 44, 45]. Each attempt takes $1\mu s$ and produces a successful entanglement with probability $p = 1/8$ [7]. After n attempts, this probability increases to $P = 1-(1-p)^n$. We set $n = 52$ to achieve $P = 99.9\%$, requiring $52\ \mu s$. The resulting Bell pairs have a fidelity of ~0.999 [7]. EPR pair generation is multi-step process that includes loading the communication qubit from the chip to an optical cavity ($100\mu s$), optical pumping and application of a $\pi/2$ pulse ($1.1\mu s$), repeated BSM ($52\mu s$), cavity de-pumping ($6\mu s$), unloading the communication qubit from the cavity to the chip ($100\mu s$), which takes $259\mu s$. However, this latency can be hidden with program operations by initiating the process ahead of time. So, if an EPR pair is required at time T, we assume that EPR pair generation starts at time t=$T-259\mu s$.

5.4 Figures of Merit

Effective Teleportation Count (T_{eff}): We compute effective teleportation count using Equation (5), where N_{RELOCATE} and $N_{\text{Re-CNOT}}$ are the number of RELOCATEs and Re-CNOTs in the schedule. As Re-CNOTs take 1.77× longer than RELOCATEs, we set α=1.77. It can be adjusted if gate characteristics change or IRIS is adopted in other technologies.

$$T_{\text{eff}} = N_{\text{RELOCATE}} + \alpha \times N_{\text{Re-CNOT}}, \tag{5}$$

Latency (L): We also measure the end-to-end latency of a program schedule, similar to prior works [22, 23].

Lower teleportation counts (T_{eff}) and latency (L) are better.

5.5 Experimental Setup

We conduct experiments on a server equipped with 160-core ARM Neoverse-N1, 512 GB RAM, and 2 TB SSD. Our software runs in Python 3.9 environment including `qiskit` [46] to parse circuits, `networkx` [47], `pymetis` [48], and `pulp` [49] for the circuit partitioning and ILP-based chip selection.

Table 5. Effective teleportation count (T_{eff}) and latency (L in seconds) comparison using different mappers on 2 × 2 DQC.

Program	Mapper-1: GCP-E				Mapper-2: sOEE				Mapper-3: WBCP				Mapper-4: Min-Cut ***(Best)***			
	QuComm		IRIS		QuComm		IRIS		QuComm		IRIS		QuComm		IRIS	
	T_{eff}	L	T_{eff}	L	T_{eff}	L	T_{eff}	L	T_{eff}	L	T_{eff}	L	T_{eff}	L	T_{eff}	L
BV	95	0.16	85	0.10	95	0.16	83	0.10	27	0.07	21	0.06	73	0.13	60	0.09
QAOA-3reg	160	0.17	139	0.07	105	0.18	83	0.07	161	0.20	123	0.07	77	0.16	57	0.07
QAOA-FC	1189	1.97	682	0.96	1551	2.09	737	0.95	987	2.35	734	1.03	906	2.11	542	0.90
QFT	1126	1.41	944	0.78	1212	1.58	930	0.78	800	1.54	697	0.81	988	1.53	656	0.84
QuGAN	283	0.54	188	0.27	93	0.34	80	0.20	141	0.39	112	0.24	36	0.26	18	0.18
QV	7182	3.35	5979	1.68	7192	3.52	6060	1.70	7244	3.36	6039	1.62	7238	3.43	5965	1.67
Shor	1430	1.95	870	0.90	1824	2.34	660	0.97	1179	2.22	602	0.92	1751	2.39	644	0.92
VQE	912	1.12	665	0.58	910	1.24	765	0.54	622	1.27	536	0.58	817	1.17	629	0.59
Relative	1	1	**0.74**	**0.50**	0.89	1.01	**0.62**	**0.49**	0.68	0.94	**0.52**	**0.47**	0.66	0.93	**0.42**	**0.47**

6 Results

6.1 Effective Teleportation Count and Latency

Table 5 shows the effective teleportation count (T_{eff}) and latency of IRIS and QuComm for different mappers on a 2×2 DQC. IRIS consistently outperforms QuComm for all mappers, showing that it is generalizable. By default, IRIS uses the Min-Cut mapper, because it outperforms the rest and dynamic mappers (WBCP and GCP-E) have their own lookahead-driven teleportation scheduling that conflicts with IRIS, eventually degrading performance. IRIS reduces T_{eff} by 36% on average and by up to 63%; and reduces latency by 0.51× on average and by up to 0.39× compared to QuComm. Table 6 compares QuComm and IRIS on 2×3 and 3×3 DQCs. IRIS reduces T_{eff} by 35% on average (up to 65%) on the 2×3 DQC and by 32% on average (up to 56%) on the 3×3 DQC. It reduces latency to 0.50× on average on the 2×3 DQC and to 0.48× on average on the 3×3 DQC, and up to 0.35× and 0.36× on 2×3 and 3×3 DQCs respectively.

Table 6. Effective teleportation count (T_{eff}) and latency (L in seconds) on 2×3 and 3×3 DQCs using Min-Cut mapping.

Program	2×3				3×3			
	QuComm		IRIS		QuComm		IRIS	
	T_{eff}	L	T_{eff}	L	T_{eff}	L	T_{eff}	L
BV	164	0.27	148	0.18	316	0.49	262	0.28
QAOA-3reg	201	0.26	153	0.11	378	0.44	289	0.16
QAOA-FC	3357	4.34	1843	1.72	7640	6.79	4802	2.94
QFT	3974	3.37	1980	1.59	7298	4.40	4022	2.52
QuGAN	89	0.43	61	0.32	186	0.66	106	0.41
QV	22496	6.28	19473	3.14	51740	9.81	44901	4.70
Shor	4642	4.90	1608	1.69	7861	6.62	3471	2.59
VQE	1998	2.12	1741	1.20	4367	3.42	4279	1.65
Relative	1.00	1.00	**0.65**	**0.50**	1.00	1.00	**0.68**	**0.48**

6.2 Impact of UMS

Figure 11 compares the cumulative T_{eff} as scheduling proceeds for a 240-qubit Shor's algorithm on a 3×3 DQC. Blocks marked by Ⓐ denote cases where IRIS selects a different chip compared to QuComm to execute the block. These selections eventually reduce teleportations in the subsequent block Ⓑ. A different scheduling for block Ⓒ reduces teleportations for all blocks beyond Block-23.

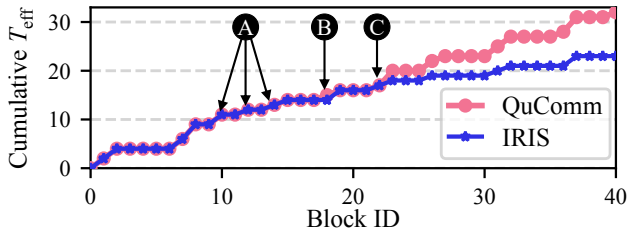

Figure 11. Cumulative teleportation count as scheduling proceeds in QuComm and IRIS for a 240-qubit Shor's algorithm. Utility-driven lookahead with multi-candidate block scheduling in IRIS reduces total teleportation costs.

6.3 Accuracy of Future T_{eff} Estimation

Recollect that IRIS estimates the costs of teleportations of a future block while scheduling. Figure 12 shows that the estimated and actual teleportation are identical costs for all blocks in a 32-qubit QAOA 3-regular program on a 2×2 DQC system. This is because relocated qubits often do not undergo teleportations, operations, or EPR-releases in between.

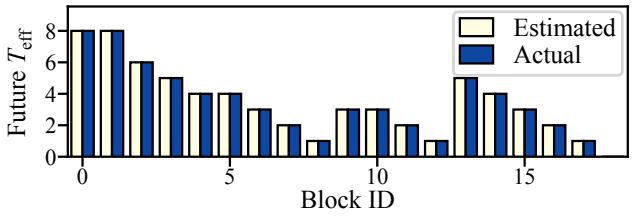

Figure 12. Estimated teleportation counts during scheduling are identical to actual costs during execution (data is obtained from a 32-qubit QAOA 3-regular program on 2 × 2 DQC).

6.4 Impact of EES

Figure 13 shows the schedule of a 240-qubit QAOA-3reg program on a 3 × 3 DQC. The schedule from QuComm takes 443 ms and has a RELOCATE concurrency of only 0.85 RELOCATE/ms. IRIS (without EES) reduces the latency to 348 ms with a RELOCATE concurrency of 0.83 RELOCATE/ms. With EES, IRIS reduces latency to 160 ms and achieves an even higher RELOCATE concurrency of 1.81 RELOCATE/ms. Also, IRIS (without EES) improves latency by 1.2× on average and up to 1.6× compared to QuComm. With EES, the latency reduces to 2.0× on average and by up to 2.9×.

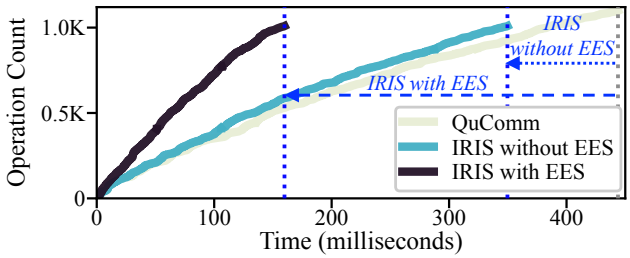

Figure 13. Schedules of a 240-qubit QAOA 3-regular program. IRIS requires fewer RELOCATEs and achieves higher RELOCATE concurrency, leading to lower program latency.

6.5 Impact of Each Optimization

Figure 14 shows the efficacy of each optimization in IRIS.

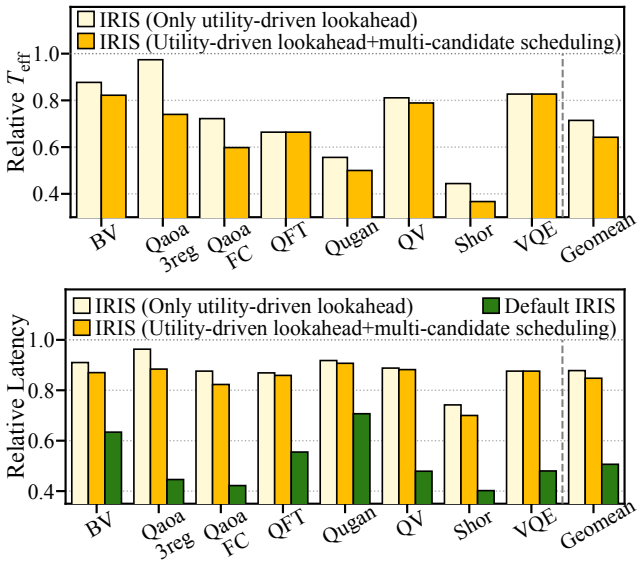

Figure 14. Impact of each optimization in IRIS on a 2×2 DQC for various programs. Values are normalized to QuComm.

Only utility-driven lookahead reduces T_{eff} and latency by 1.40× and 1.14× on average, respectively. Introducing multi-candidate scheduling on top further reduces T_{eff} and latency by 1.57× and 1.18×. Finally, introducing EES reduces latency by 1.98× on average compared with QuComm. Note that both utility-driven lookahead and multi-candidate scheduling reduce both T_{eff} and latency. EES does not reduce T_{eff}, but only improves latency by making a schedule more compact.

We also compare the default IRIS against multiple variants of QuComm and IRIS, as shown in Figure 15, for a 120-qubit QAOA 3-regular program on a 2 × 2 DQC. For QuComm, we consider next-3 blocks for lookahead, besides the default. For IRIS, we consider three variants: (a)-with one candidate and utility-driven lookahead, (b)-next-3 blocks for lookahead with multi-candidate scheduling, and (c)-default (both utility-driven lookahead and multi-candidate scheduling). QuComm with next-3 blocks lookahead is ineffective. Similarly, multi-candidate scheduling outperforms single candidate variants.

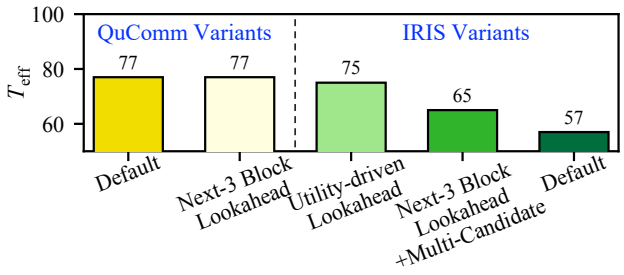

Figure 15. Effective teleportation count of a 120-qubit QAOA 3-regular graph with different QuComm and IRIS variants.

6.6 Impact of Scaling System Sizes

Figure 16(a) evaluates IRIS across four chip sizes on a 2×2 DQC using QAOA 3-regular programs. The number of qubits per chip ranges from 40 to 160 and program sizes range from 120 to 500 qubits. IRIS reduces T_{eff} by 18–22% and latency by 60–64%. While IRIS consistently outperforms QuComm for all configurations, we cannot conclude on any scaling trends for T_{eff} because the programs differ in sizes. The latency improves with increasing chip (and program) sizes because IRIS has more opportunities to optimize the schedules. Figure 16(b) shows the impact of increasing the number of chips. IRIS consistently lowers T_{eff} and latency.

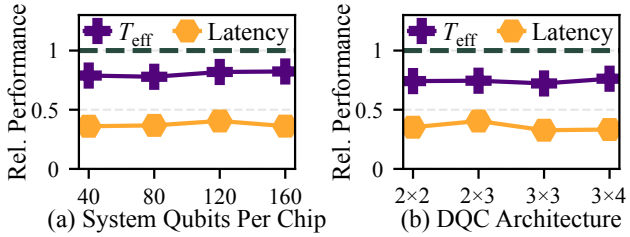

Figure 16. Impact of scaling on IRIS performance using QAOA 3-regular programs relative to QuComm. (a) Scaling the chip size on a 2×2 DQC. (b) Scaling the number of chips.

6.7 Sensitivity to Design Parameters

Figure 17 shows the impact of increasing the number of solutions in the tree (w) and scheduling group sizes ($|G|$) on the performance and runtime of IRIS. While increasing w initially improves the performance, the returns diminish beyond w=16. Increasing group sizes does not significantly improve performance beyond $|G|$=4. Increasing both leads to increased compilation time which is expected due to increased search space. By default, IRIS uses (w,$|G|$)=(16,4).

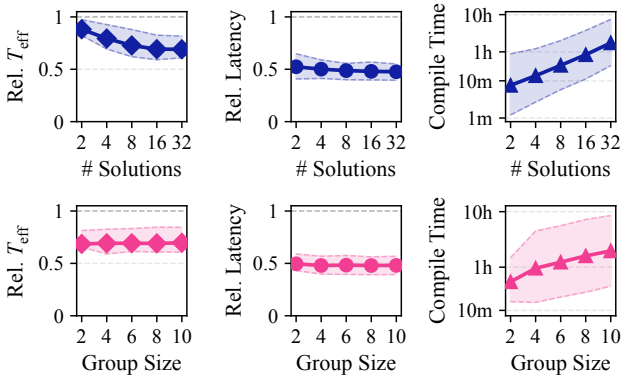

Figure 17. Impact of w and $|G|$ on UMS. Values are relative to QuComm based on average from over 24 benchmarks.

6.8 Sensitivity to Hardware Parameters

Re-CNOT to RELOCATE Cost Ratio (α). The Re-CNOT to RELOCATE cost ratio depends on the hardware. Figure 18 shows the teleportation count of a QAOA 3-regular program on 2×2 DQC relative to QuComm for various Re-CNOT to RELOCATE cost ratios. IRIS consistently reduces T_{eff}.

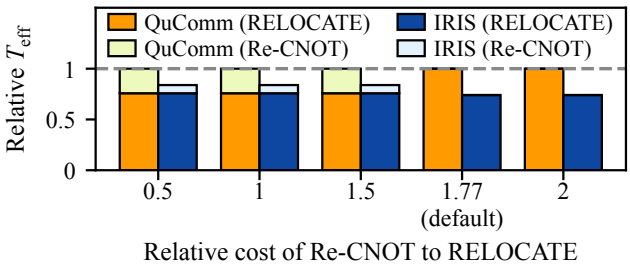

Figure 18. IRIS performance relative to QuComm for varying Re-CNOT to RELOCATE cost ratios.

Percentage of Communication Qubits. By default, IRIS reserves 10% of qubits per chip for communication because non-local CNOT count is often much lower than local CNOTs. Figure 19 compares the performance of a 120-qubit QAOA 3-regular program on 2×2 DQC with increasing percentages of communication qubits. IRIS consistently outperforms.

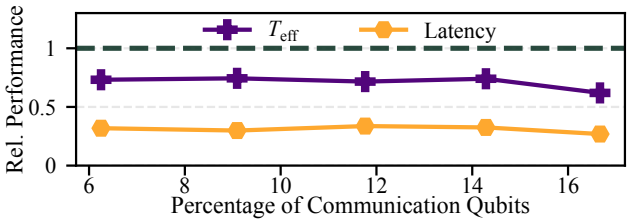

Figure 19. Performance comparison using a 120-qubit QAOA 3-regular program on 2×2 DQC across various percentages of communication qubits. Values are normalized to QuComm.

6.9 Impact of EPR Pair Generation Latency

We assume EPR pair generation can be fully overlapped with atom movements, but hardware limitations may sometimes prevent this. For example, Rb atoms take 2.67× longer for EPR pair generation than Yb atoms [7, 50]. Figure 20 shows the latency of a 240-qubit QAOA-FC on a 3×3 DQC as EPR pair generation latency hidden varies from 0% (no hiding) to 100% (fully hidden). IRIS consistently outperforms QuComm and is more advantageous when this latency cannot be hidden, as its schedule has fewer teleportations and higher concurrency.

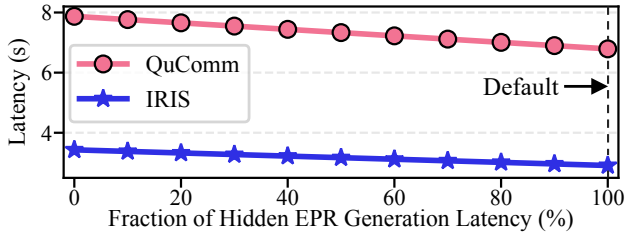

Figure 20. Latency of a 240-qubit QAOA-FC program on 3×3 DQC with increasing fraction of hidden EPR generation latency. IRIS consistently outperforms QuComm.

6.10 Impact on Quantum Error Correction (QEC)

Fault-tolerant quantum computers map program variables to logical qubits encoded in quantum error correction (QEC) codes and translate instructions to logical operations first. These operations are then translated into low-level hardware compatible instructions (similar to IRIS). IRIS can be integrated into existing compilers that focus on logical synthesis, but developing an end-to-end fault-tolerant DQC compiler is beyond the scope of our paper. To show that optimized low-level scheduling can still benefit fault-tolerant systems, we compare the scheduling of IRIS against QuComm for three promising QEC codes on a 2×2 DQC.

As shown in Table 7, IRIS reduces T_{eff} by 9–42% and logical cycle time (in milliseconds) by 51–65% compared to QuComm. Logical cycle time is the time taken to extract parity information by running a syndrome extraction circuit on data qubits (that store information) and parity qubits (that

detect errors) over multiple rounds. It dictates the runtime of programs. In surface code [51] and color code [52], each parity qubit interacts with only up to four data qubits, resulting in fewer non-local CNOTs than in bivariate bicycle code [53], where a parity qubit interacts with up to six data qubits and there is a greater likelihood that the data qubits are spread across multiple chips. The costs may be further reduced with QEC-specific mappers [54]. We leave it for future work.

Table 7. Performance of a logical cycle on a 2×2 DQC.

Code	QuComm		IRIS	
	T_{eff}	L_{cycle}	T_{eff}	L_{cycle}
Bivariate Bicycle [[72,12,6]]	2566	1192	2337	588
Color [[61,1,9]]	1106	1878	797	722
Surface [[49,1,7]]	589	738	343	257
Relative	1.00	1.00	**0.73**	**0.40**

7 Complexity Analysis

We analyze IRIS's complexity assuming a maximum block size B. *First*, BFA searches up to B gates and fuses blocks by again searching up to B gates per iteration. Over an entire program, with at most $\frac{V}{B}$ blocks, BFA has a complexity of $C_{\text{BFA}} = O(V \cdot B)$. *Second*, UMS schedules one block at a time while preserving up to w solutions. To schedule each non-local gate, UMS considers up to N chips. If EPR release is needed, UMS additionally considers which qubit to move and to which chip. Thus, the complexity of scheduling a non-local gate is $C_{\text{gate}} = O(N \log N + N \cdot n_{\text{qubit}})$. For the block's i^{th} non-local gate, UMS evaluates at most $|G| \cdot (B - i)$ future gates in the scheduling group. As a block contains at most B non-local gates and UMS preserves up to w solutions, the complexity of UMS per block is $C_{\text{UMS}} = O(w \cdot |G| \cdot B^2 \cdot C_{\text{gate}})$. IRIS iterates UMS until all V program gates are scheduled. The number of iterations is at most $\frac{V}{B}$. *Lastly*, EES identifies the set of operations whose data dependencies are resolved, which takes $|G| \cdot B$. For these operations, EES linearly searches the previous operations, and its complexity is $C_{\text{EES}} = O(|G|^2 \cdot B^2)$. The overall complexity of IRIS is given in Equation (6).

$$C_{\text{IRIS}} = O(VB[w \cdot |G| \cdot N \cdot (\log N + n_{\text{qubit}}) + |G|^2]) \quad (6)$$

When a block grows up to the entire program (worst case), the IRIS complexity scales quadratic in the program size. For a fixed maximum block size (typical case), solutions and group size, the complexity grows linearly with program size.

Scalability and Performance Comparison: We compare the complexity and performance of IRIS and QuComm for QAOA 3-regular programs with three configurations. Table 8 shows the memory requirement, compiler runtime, effective teleportation count, and program latency for IRIS relative to QuComm. Although IRIS takes longer than QuComm to compile, its schedule has much lower execution time which leads to a reduced end-to-end program runtime because quantum programs run for thousands of shots. For example, compiling a 1100-qubit program with 3.3K CNOTs using QuComm takes 2 minutes and produces a schedule of 785ms latency. Assuming 50K shots, the total runtime of the program is 2 minutes + 50K × 785 ms = 656 minutes. In contrast, IRIS takes 150 minutes to compile but produces a schedule with 307 ms latency, leading to a total runtime of 150 minutes + 50K × 307 ms = 406 minutes, 38% lower than QuComm.

Table 8. Complexity analysis and performance of IRIS relative to QuComm for a program executing 50K shots.

DQC Arch	Program Qubits	Memory	T_{eff}	Compile Time	Schedule Latency	Program Runtime
2×2	500	2.08×	0.83×	363×	0.38×	0.80×
2×3	800	2.89×	0.83×	201×	0.35×	0.61×
3×3	1100	2.53×	0.85×	74×	0.39×	0.62×

8 Related Work

We compare and contrast IRIS with prior DQC compilers.

DQC Mappers. DQC mappers assign program qubits to compute qubits to minimize non-local gates. Static partitioning [28, 55–57] computes a single assignment at compile time. Dynamic partitioning [30, 31, 54, 58, 59] instead divides a program into segments and reconfigures qubit layout per segment, incurring extra RELOCATEs at segment boundaries for fewer non-local gates within each segment. These mappers provide better qubit layout but leave non-local gates for the scheduler. IRIS focuses on instruction scheduling and is orthogonal to mapping. IRIS is compatible with both types of mapping policies and outperforms consistently.

DQC Schedulers. Ferrari et al. [21] independently schedule each non-local gate. AutoComm groups non-local gates that share a qubit and span two chips into a block [22]. QuComm [23] extends blocks across multiple chips under EPR capacity. These methods have limited lookahead capabilities and IRIS outperforms them. Cuomo et al. [19] formulate non-local gate scheduling as an ILP, but support only Re-CNOTs. As Re-CNOT performs a CNOT between two chips without relocating any qubit, each qubit stays in its initial chip for the program. Thus, this compiler misses opportunities to reduce teleportations through qubit displacements. IRIS overcomes this drawback. Promponas et al. [60] use deep reinforcement learning for scheduling, but their method does not scale beyond 50 gates. In contrast, IRIS scales to large programs. AdaptDQC adopts circuit cutting for non-local gate scheduling [61]. However, the post-processing cost of circuit cutting grows exponentially in the number of non-local gates, limiting scalability. SwitchQNet [62] reduces dynamic switching overheads in the all-to-all connected DQCs [63–66] rather than teleportations, which is orthogonal to IRIS.

9 Conclusion

We propose *IRIS*, a compiler that reduces teleportation costs in Distributed Quantum Computers (DQCs). *IRIS* employs two features. *First, Utility-Driven Lookahead with Multi- Candidate Block Scheduling (UMS)* to schedule operations by (1) using a utility-driven lookahead window that only considers useful future blocks and (2) retaining multiple candidate schedules to prevent early commitment to sub-optimal schedules. *Second*, IRIS employs *EPR-Capacity-Aware Early Scheduling (EES)* to exploit available EPR resources to initiate the execution of future gates and their teleportations early. Our evaluations show that IRIS reduces teleportations by 34% on average and by up to 65%, while reducing latency by 2.0× on average and by up to 2.9× compared to the baseline.

Acknowledgments

We thank Hezi Zhang for her help in implementing QuComm on our end. We thank Avinash Kumar, Sourish Wawdhane, Ravi Ghadia, Dongwhee Kim, and Reza Nejabati for generic discussions. We also thank the reviewers of EuroSys 2027 for their feedback and our shepherd for helping us improve the paper. Last but not the least, we thank the generous support from Cisco and Cockrell School of Engineering at UT Austin.

A EPR Pair Generation Process

We explain EPR generation using Figure 21, based on the protocol by Li et al. [7]. A non-local CNOT requires an EPR pair, qubit movements, and local operations. To perform a CNOT between qubits 0 and 1, communication qubits a and b form an EPR pair. Qubits 0 and 1 move to the interaction range of a and b respectively, followed by CNOT 0, a and CNOT b, 1 to form an EPR pair. The communication qubits are loaded into cavities and optically pumped. The photons emitted from the spin-up ($|\uparrow\rangle$) or spin-down state ($|\downarrow\rangle$) are directed to Bell state measurement (BSM) to check for a successful entanglement. This process is probabilistic with a success rate, P, of 12.5%. To produce an EPR pair with a fidelity of 99.9%, we use a repeat until success approach with N attempts. The probability of successful entanglement after N attempts is given by $(1\text{-}(1\text{-}P)^N)$. For it to be 99.9%, we need at least 52 attempts. Next, the qubits are depumped and moved out of the cavities. Based on device studies [7, 67], EPR pair generation takes 259µs while moving atoms takes 300µs. This allows us to fully hide the EPR generation latency.

B Protocol for Non-Local CNOTs

We explain the detailed execution of non-local CNOTs in neutral-atom DQCs. Figure 22(a) shows a RELOCATE. Qubit 0 moves within the interaction range of qubit a and CNOT 0, a is executed. State teleportation is used during which qubits a and b collapse after measurement. CNOT 0, 1 is performed by bringing qubit 1 within interaction range of qubit 0. Figure 22(b) shows a Re-CNOT operation. Qubit 0 is

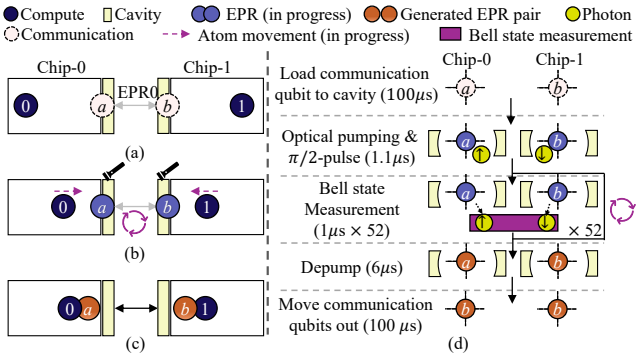

Figure 21. Overview of EPR pair generation.

moved within the interaction range of qubit a and CNOT 0, a is executed. Gate teleportation is used here which measures qubit a and prepares qubit b as a proxy for the remote CNOT. Next, qubit 1 is moved within the interaction range of b, CNOT b,1 is executed, and b collapses after measurement.

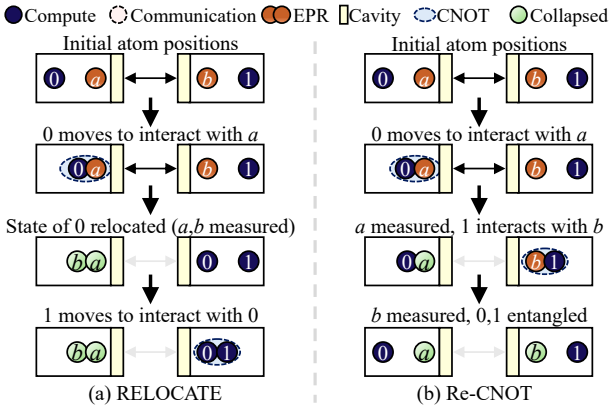

Figure 22. Implementation of the non-local CNOT.

C IRIS Algorithm

Algorithm 1 gives an overview of the IRIS algorithm.

Algorithm 1 IRIS

```
Input: Program, Qubit Mapping (M), EPR Capacity (D)
Output: Schedule (Sched)
function IRIS(Program, M, D)
    DAG ← Build_DAG(Program)
    B ← Block_Formation(DAG, M, D)
    T ← Initialize_Solution_Tree(M, D)
    while not all blocks in B are scheduled do
        C ← next unscheduled block in B
        G ← Utility_Driven_Lookahead_Window(C, B)
        T ← UMS(C, G, T, M, D)
    Sched ← Best_Schedule(T)
    Sched ← EES(Sched)
    return Sched
```

References

[1] James Ang, Gabriella Carini, Yanzhu Chen, Isaac Chuang, Michael Demarco, Sophia Economou, Alec Eickbusch, Andrei Faraon, Kai-Mei Fu, Steven Girvin, Michael Hatridge, Andrew Houck, Paul Hilaire, Kevin Krsulich, Ang Li, Chenxu Liu, Yuan Liu, Margaret Martonosi, David McKay, Jim Misewich, Mark Ritter, Robert Schoelkopf, Samuel Stein, Sara Sussman, Hong Tang, Wei Tang, Teague Tomesh, Norm Tubman, Chen Wang, Nathan Wiebe, Yongxin Yao, Dillon Yost, and Yiyu Zhou. Arquin: Architectures for multinode superconducting quantum computers. *ACM Transactions on Quantum Computing*, 5(3), September 2024.

[2] Junyu Liu, Connor T. Hann, and Liang Jiang. Data centers with quantum random access memory and quantum networks. *Phys. Rev. A*, 108:032610, Sep 2023.

[3] David Awschalom, Karl K. Berggren, Hannes Bernien, Sunil Bhave, Lincoln D. Carr, Paul Davids, Sophia E. Economou, Dirk Englund, Andrei Faraon, Martin Fejer, Saikat Guha, Martin V. Gustafsson, Evelyn Hu, Liang Jiang, Jungsang Kim, Boris Korzh, Prem Kumar, Paul G. Kwiat, Marko Lončar, Mikhail D. Lukin, David A.B. Miller, Christopher Monroe, Sae Woo Nam, Prineha Narang, Jason S. Orcutt, Michael G. Raymer, Amir H. Safavi-Naeini, Maria Spiropulu, Kartik Srinivasan, Shuo Sun, Jelena Vučković, Edo Waks, Ronald Walsworth, Andrew M. Weiner, and Zheshen Zhang. Development of quantum interconnects (quics) for next-generation information technologies. *PRX Quantum*, 2:017002, Feb 2021.

[4] R. Van Meter, K. Nemoto, W. Munro, and K. Itoh. Distributed arithmetic on a quantum multicomputer. In *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA)*, pages 354–365, 2006.

[5] Eun Oh, Xuanying Lai, Jianming Wen, and Shengwang Du. Distributed quantum computing with photons and atomic memories. *Advanced Quantum Technologies*, 6(6):2300007, 2023.

[6] Nitish K. Chandra, Eneet Kaur, and Kaushik P. Seshadreesan. Network operations scheduling for distributed quantum computing. In *2024 IEEE 6th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA)*, pages 506–515, 2024.

[7] Yiyi Li and Jeff D. Thompson. High-rate and high-fidelity modular interconnects between neutral atom quantum processors. *PRX Quantum*, 5:020363, Jun 2024.

[8] D Main, P Drmota, DP Nadlinger, EM Ainley, A Agrawal, BC Nichol, R Srinivas, G Araneda, and DM Lucas. Distributed quantum computing across an optical network link. *Nature*, pages 1–6, 2025.

[9] 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. *Phys. Rev. A*, 89:022317, Feb 2014.

[10] Almudena Carrera Vazquez, Caroline Tornow, Diego Ristè, Stefan Woerner, Maika Takita, and Daniel J. Egger. Combining quantum processors with real-time classical communication. *Nature*, 2024.

[11] Y.-C. Wei, P.-J. Stas, A. Suleymanzade, G. Baranes, F. Machado, Y. Q. Huan, C. M. Knaut, S. W. Ding, M. Merz, E. N. Knall, U. Yazlar, M. Sirotin, I. W. Wang, B. Machielse, S. F. Yelin, J. Borregaard, H. Park, M. Lončar, and M. D. Lukin. Universal distributed blind quantum computing with solid-state qubits. *Science*, 2025.

[12] IonQ. IonQ's Accelerated Roadmap: Turning Quantum Ambition into Reality. https://ionq.com/blog/ionqs-accelerated-roadmap-turning-quantum-ambition-into-reality, 2024. Accessed: 2025-06-27.

[13] Pasqal. Pasqal Technology Roadmap 2025. https://www.pasqal.com/technology/roadmap/, 2025. Accessed: 2025-06-27.

[14] IBM. The ibm quantum development roadmap. https://www.ibm.com/quantum/roadmap.

[15] H Aghaee Rad, T Ainsworth, RN Alexander, B Altieri, MF Askarani, R Baby, L Banchi, BQ Baragiola, JE Bourassa, RS Chadwick, et al. Scaling and networking a modular photonic quantum computer. *Nature*, 638(8052):912–919, 2025.

[16] Charles H. Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K. Wootters. Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels. *Phys. Rev. Lett.*, 70:1895–1899, Mar 1993.

[17] Marcello Caleffi, Michele Amoretti, Davide Ferrari, Jessica Illiano, Antonio Manzalini, and Angela Sara Cacciapuoti. Distributed quantum computing: A survey. *Computer Networks*, 254:110672, 2024.

[18] D. Bluvstein, S. J. Evered, A. A. Geim, et al. Logical quantum processor based on reconfigurable atom arrays. *Nature*, 626:58–65, 2024.

[19] Daniele Cuomo, Marcello Caleffi, Kevin Krsulich, Filippo Tramonto, Gabriele Agliardi, Enrico Prati, and Angela Sara Cacciapuoti. Optimized compiler for distributed quantum computing. *ACM Transactions on Quantum Computing*, 4(2), February 2023.

[20] Longshan Xu, Edwin Hsing-Mean Sha, Xiulin Cui, and Qingfeng Zhuge. Optimizing quantum circuit mapping to reduce inter-module communications in distributed architectures. In *SC*. ACM, 2025.

[21] Davide Ferrari, Angela Sara Cacciapuoti, Michele Amoretti, and Marcello Caleffi. Compiler design for distributed quantum computing. *IEEE Transactions on Quantum Engineering*, 2:1–20, 2021.

[22] Anbang Wu, Hezi Zhang, Gushu Li, Alireza Shabani, Yuan Xie, and Yufei Ding. Autocomm: A framework for enabling efficient communication in distributed quantum programs. In *55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022.

[23] Anbang Wu, Yufei Ding, and Ang Li. Qucomm: Optimizing collective communication for distributed quantum computing. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '23, page 479–493, New York, NY, USA, 2023. Association for Computing Machinery.

[24] Kaitlin N. Smith, Gokul Subramanian Ravi, Jonathan M. Baker, and Frederic T. Chong. Scaling superconducting quantum computers with chiplet architectures. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1092–1109, 2022.

[25] Hezi Zhang, Keyi Yin, Anbang Wu, Hassan Shapourian, Alireza Shabani, and Yufei Ding. Mech: Multi-entry communication highway for superconducting quantum chiplets. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 699–714, New York, NY, USA, 2024. Association for Computing Machinery.

[26] Brandon Grinkemeyer, Elmer Guardado-Sanchez, Ivana Dimitrova, Danilo Shchepanovich, G. Eirini Mandopoulou, Johannes Borregaard, Vladan Vuletić, and Mikhail D. Lukin. Error-detected quantum operations with neutral atoms mediated by an optical cavity, 2024.

[27] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for NISQ-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1001–1014, 2019.

[28] Taehoon Park and Chae Y. Lee. Algorithms for partitioning a graph. *Computers & Industrial Engineering*, 28(4):899–909, 1995.

[29] Wikipedia contributors. Minimum cut. https://en.wikipedia.org/wiki/Minimum_cut, 2024. Accessed: 2025-06-27.

[30] E. Kaur, S. Pouryousef, H. Shapourian, J. Zhao, M. Kilzer, R. Kompella, and R. Nejabati. Optimized quantum circuit partitioning across multiple quantum processors. *IEEE Transactions on Quantum Engineering*, 6:1–17, 2025.

[31] F. Burt, K.-C. Chen, and K.K. Leung. Generalised circuit partitioning for distributed quantum computing. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2024.

[32] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 11–20, 1993.

[33] Ernst Ising. Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik*, 31(1):253–258, 1925.

[34] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014.

[35] Don Coppersmith. An approximate fourier transform useful in quantum factoring. *arXiv preprint quant-ph/0201067*, 2002. Quantum Fourier Transform reference.

[36] Andrew W. Cross, Lev S. Bishop, Sarah Sheldon, Paul D. Nation, and Jay M. Gambetta. Validating quantum computers using randomized model circuits. *Physical Review A*, 100(3):032328, 2019. Quantum Volume benchmark.

[37] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5:4213, 2014.

[38] PsiQuantum team. A manufacturable platform for photonic quantum computing. *Nature*, 641:876–883, 2025.

[39] Wan-Hsuan Lin, Daniel Bochen Tan, and Jason Cong. Reuse-aware compilation for zoned quantum architectures based on neutral atoms. In *Proceedings of the 2025 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1–14, Las Vegas, NV, USA, March 2025. IEEE. Accepted to HPCA 2025.

[40] Jixuan Ruan, Xiang Fang, Hezi Zhang, Ang Li, Travis Humble, and Yufei Ding. Powermove: Optimizing compilation for neutral atom quantum computers with zoned architecture. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '25, page 163–178, New York, NY, USA, 2025. Association for Computing Machinery.

[41] Zhanchuan Zhang, Jeth Arunseangroj, and Wenchao Xu. Dual-type dual-element atom arrays for quantum information processing. *arXiv preprint arXiv:2503.16896*, 2025.

[42] Shraddha Anand, Conor E. Bradley, Ryan White, Vikram Ramesh, Kevin Singh, and Hannes Bernien. A dual-species rydberg array. *Nature Physics*, 20:1744–1750, 2024.

[43] Tzu Ken Shen, Yi Zhu, Haley Nguyen, Nicholas Lyu, Binhan Hua, and Giulia Semeghini. Dual species rb-yb atom array. In *APS Division of Atomic, Molecular and Optical Physics Meeting 2024*, Fort Worth, Texas, June 2024. American Physical Society. Abstract id. K00.100.

[44] Jian-Wei Pan, Christoph Simon, Časlav Brukner, and Anton Zeilinger. Entanglement purification for quantum communication. *Nature*, 410:1067–1070, 2001.

[45] Charles H. Bennett, Gilles Brassard, Sandu Popescu, Benjamin Schumacher, John A. Smolin, and William K. Wootters. Purification of noisy entanglement and faithful teleportation via noisy channels. *Physical Review Letters*, 76(5):722–725, 1996.

[46] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J. Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D. Nation, Lev S. Bishop, Andrew W. Cross, Blake R. Johnson, and Jay M. Gambetta. Quantum computing with qiskit, 2024.

[47] NetworkX Developers. Networkx. https://networkx.org/.

[48] Andreas Klöckner. Pymetis. https://github.com/inducer/pymetis.

[49] Stuart Mitchell, Michael O'Sullivan, and Iain Dunning. PuLP: A linear programming toolkit for python. Technical report, The University of Auckland, Auckland, New Zealand, September 2011.

[50] Ziv Aqua, Matthew L Peters, David C Spierings, Guoqing Wang, Edita Bytyqi, Thomas Propson, and Vladan Vuletić. Mode multiplexing for scalable cavity-enhanced operations in neutral-atom arrays. *arXiv preprint arXiv:2511.20858*, 2025.

[51] A.Yu. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003.

[52] H. Bombin and M. A. Martin-Delgado. Topological quantum distillation. *Phys. Rev. Lett.*, 97:180501, Oct 2006.

[53] Sergey Bravyi, Andrew W. Cross, Jay M. Gambetta, Dmitri Maslov, Patrick Rall, and Theodore J. Yoder. High-threshold and low-overhead fault-tolerant quantum memory. *Nature*, 627(8005):778–782, 2024.

[54] Peter Wegmann, Aleksandra Świerkowska, Emmanouil Giortamis, and Pramod Bhatotia. Chipmunq: A fault-tolerant compiler for chiplet quantum architectures, 2026.

[55] O. Daei, K. Navi, and M. Zomorodi-Moghadam. Optimized quantum circuit partitioning. *International Journal of Theoretical Physics*, 59(12):3804–3820, December 2020.

[56] Waldemir Cambiucci, Regina M. Silveira, and Wilson V. Ruggiero. Hypergraphic partitioning of quantum circuits for distributed quantum computing. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 02, pages 217–227, 2023.

[57] R. G. Sundaram, H. Gupta, and C. R. Ramakrishnan. Efficient distribution of quantum circuits. In *35th International Symposium on Distributed Computing (DISC 2021)*, 2021.

[58] E. Nikahd, N. Mohammadzadeh, M. Sedighi, and M. S. Zamani. Automated window-based partitioning of quantum circuits. *Physica Scripta*, 96(3):035102, January 2021.

[59] Jonathan M. Baker, Casey Duckering, Alexander Hoover, and Frederic T. Chong. Time-sliced quantum circuit partitioning for modular architectures. In *Proceedings of the 17th ACM International Conference on Computing Frontiers*, CF '20, page 98–107, New York, NY, USA, 2020. Association for Computing Machinery.

[60] P. Promponas, A. Mudvari, L. Della Chiesa, P. Polakos, L. Samuel, and L. Tassiulas. Compiler for distributed quantum computing: a reinforcement learning approach. In *ICC 2025-IEEE International Conference on Communications*, pages 4615–4621. IEEE, 2025.

[61] D. Xiang, L. Lu, S. Tan, X. Jia, Z. Zhou, G. Sun, M. Chen, and J. Yin. Adaptdqc: Adaptive distributed quantum computing with quantitative performance analysis. *IEEE Transactions on Computers*, 2025.

[62] Hezi Zhang, Yiran Xu, Haotian Hu, Keyi Yin, Hassan Shapourian, Jiapeng Zhao, Ramana Rao Kompella, Reza Nejabati, and Yufei Ding. Switchqnet: Optimizing distributed quantum computing for quantum data centers with switch networks. In *Proceedings of 49th International Symposium on Computer Architecture (ISCA)*. IEEE/ACM, 2025.

[63] Jiapeng Zhao, Stéphane Vinet, Amir Minoofar, Michael Kilzer, Lucas Wang, Galan Moody, Vijoy Pandey, Ramana Kompella, and Reza Nejabati. A universal quantum information preserving photonic switch for scalable quantum networks, 2026.

[64] Mahdi Bornadel, Stéphane Vinet, Jiapeng Zhao, Amir Minoofar, Michael Kilzer, Ramana Kompella, and Reza Nejabati. Towards a quantum switch preserving polarization entanglement. In Philip R. Hemmer, Alan L. Migdall, and Ivan A. Burenkov, editors, *Quantum Computing, Communication, and Simulation VI*, volume 13919, page 139190F. International Society for Optics and Photonics, SPIE, 2026.

[65] Eneet Kaur, Hassan Shapourian, Jiapeng Zhao, Michael Kilzer, Shahrooz Pouryousef, Amir Minoofar, Mahdi Bornadel, Ramana Kompella, and Reza Nejabati. Quantum data centers: analysis of different architectural solutions. In Philip R. Hemmer, Alan L. Migdall, and Ivan A. Burenkov, editors, *Quantum Computing, Communication, and Simulation VI*, volume PC13919, page PC1391908. International Society for Optics and Photonics, SPIE, 2026.

[66] Jiapeng Zhao, Yang Xu, Xiyuan Lu, Eneet Kaur, Michael Kilzer, Ramana Kompella, Robert W. Boyd, and Reza Nejabati. Scalable low-latency entanglement distribution for distributed quantum computing. *Optica Quantum*, 3(6):606–616, Dec 2025.

[67] Dolev Bluvstein, Harry Levine, Giulia Semeghini, Tout T Wang, Sepehr Ebadi, Marcin Kalinowski, Alexander Keesling, Nishad Maskara, Hannes Pichler, Markus Greiner, et al. A quantum processor based on coherent transport of entangled atom arrays. *Nature*, 604(7906):451–456, 2022.

D Artifact Appendix

D.1 Abstract

This artifact comprises a codebase and dataset. The codebase contains the DQC mappers and DQC schedulers. For the dataset, benchmark circuits, DQC mapping, the generated DQC schedules, and the raw results reported in this paper are included. The authors provide two reproduction workflows: a *data-only* workflow is for the evaluators who use a small machine. It provides the paper's tables and figures directly from the IRIS dataset within an hour. A *full* workflow that re-runs all compilations and reproduces the paper results in about 7 days on a 160-core server.

D.2 Artifact check-list (meta-information)

- **Algorithm:** QuComm (baseline) and IRIS (ours).
- **Program:** Eight benchmarks (BV, QAOA-3reg, QAOA-FC, QFT, QuGAN, QV, Shor, and VQE) and three QEC codes (Bivariate Bicycle, Color, and Surface); all circuits are included.
- **Data set:** IRIS-dataset (1.4 GB archive on Zenodo) contains benchmark circuits, mapper outputs, DQC schedules, and the raw results.
- **Run-time environment:** Linux; Python 3.9 Conda environment (`environment.yml` provided).
- **Hardware:** Any Linux machines (≥16 GB RAM) for the data-only workflow; a multi-core server is recommended for the full re-run (we use 160-cores machine).
- **Metrics:** Effective teleportation count and program latency.
- **Output:** CSV/LaTeX tables and PDF figures matching the paper.
- **How much disk space required (approximately)?** About 40 GB (1.4 GB dataset; the rest for the full re-run).
- **How much time is needed to prepare workflow (approximately)?** Less than one hour.
- **How much time is needed to complete experiments (approximately)?** Data-only workflow takes under one hour, while full re-run takes about 7 days on a 160-core machine.
- **Publicly available?** Yes, codebase is available.
- **Code licenses (if publicly available)?** MIT.
- **Data licenses (if publicly available)?** MIT.
- **Archived (provide DOI)?** 10.5281/zenodo.22152933 (Zenodo).

D.3 Description

1) How to access: The codebase, mapping generated by the mappers, schedule generated by the schedulers, benchmark programs and the results of raw data are available on Zenodo.

2) Hardware dependencies: No special hardware and no GPU is required. The data-only workflow runs on any Linux/macOS machines with ≥16 GB RAM. For the full re-run, the authors recommend using a compute cluster or a set of servers, because the time required for reproduction grows linearly in the number of cores. The authors use a 160-core machine.

3) Software dependencies: Conda (Miniforge/Anaconda). A `setup.sh` file creates the `iris-ae` Python environment from the environment file `environment.yml`.

4) Data sets: is archived on Zenodo. `IRIS-dataset.tar` file contains the benchmark circuits, DQC mappings, the generated schedule, and the raw results.

D.4 Installation

Run the following commands to set up:

```
bash setup.sh && conda activate iris-ae
```

D.5 Experiment workflow

The artifact supports two workflows that share the same output format.

1) Data-only (Less than an hour): This workflow regenerates every table and figure directly from the released schedules and results, without re-running the compiler:

```
bash scripts/get_data_all.sh --from_dataset
```

2) Full re-run (~7 days on 160 cores): This workflow rerun every scheduling experiment from the released mappings, then regenerate the tables and figures from the new results:

```
bash scripts/reproduce_all.sh
bash scripts/get_data_all.sh --from_results
```

D.6 Evaluation and expected results

The workflows generate experimental results. The tables are written as CSV/Latex files under `data_generator/output/` and the figures as PDFs under `figures/`. The workflows reproduce the following tables and figures:

- **Tables:** 1, 2, 5, 6, 7, and 8.
- **Figures:** 11, 12, 13, 14, 15, 16, 17, 18, 19, and 20.

Every reported trend and relative improvement are expected to match the reported results; compile times may differ across execution environments (library versions and machines).

D.7 Experiment customization

Experiments can be customized with command-line arguments. For example, a single (benchmark, architecture, mapper) combination can be run with the following line:

```
bash scripts/run_all_variants.sh bv_n120 F120 ILP
```

In addition, custom architectures can be specified as `S<qubits/chip>C<EPR/link>-<rows>x<cols>`, and a new benchmark can be added as OpenQASM 2 files under `bench/`.

D.8 Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-and-badging-current
- https://cTuning.org/ae