2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)

MIRAGE: Quantum Circuit Decomposition and Routing Collaborative Design using Mirror Gates

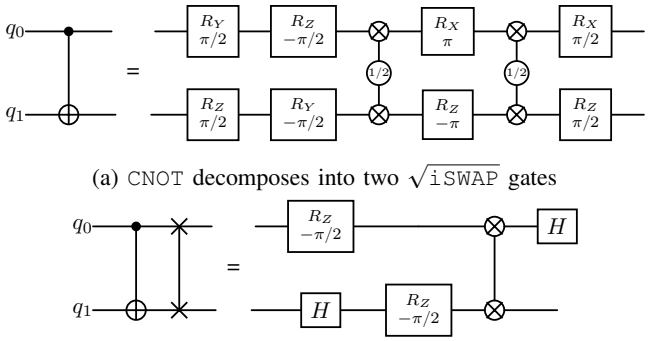
Evan McKinney†, Michael Hatridge‡, Alex K. Jones†
Department of Electrical and Computer Engineering†, Department of Physics and Astronomy‡,
University of Pittsburgh
evm33@pitt.edu, hatridge@pitt.edu, akjones@pitt.edu

Abstract*—Building efficient large-scale quantum computers is a significant challenge due to limited qubit connectivities and noisy hardware operations. Transpilation is critical to ensure that quantum gates are on physically linked qubits, while minimizing `SWAP` gates and simultaneously finding efficient decomposition into native *basis gates*. The goal of this multifaceted optimization step is typically to minimize circuit depth and to achieve the best possible execution fidelity. In this work, we propose *MIRAGE*, a collaborative design and transpilation approach to minimize `SWAP` gates while improving decomposition using *mirror gates*. Mirror gates utilize the same underlying physical interactions, but when their outputs are reversed, they realize a different or *mirrored* quantum operation. Given the recent attention to $\sqrt{\texttt{iSWAP}}$ as a powerful basis gate with decomposition advantages over `CNOT`, we show how systems that implement the `iSWAP` family of gates can particularly benefit from mirror gates. Further, *MIRAGE* uses mirror gates to reduce routing pressure and reduce true circuit depth instead of just minimizing `SWAP`s. We explore the benefits of decomposition for $\sqrt{\texttt{iSWAP}}$ and $\sqrt[4]{\texttt{iSWAP}}$ using mirror gates, including both expanding Haar coverage and conducting a detailed fault rate analysis trading off circuit depth against approximate gate decomposition. We also describe a novel greedy approach accepting mirror substitution at different aggression levels within MIRAGE. For `iSWAP` systems that use square-lattice topologies, *MIRAGE* provides an average of 29.6% reduction in circuit depth by eliminating an average of 59.9% `SWAP` gates, with a relative decrease in infidelity of 28%. MIRAGE also improves circuit depth and decreases relative infidelity by 25% and 21% for `CNOT`-based and 23% and 19% `SYC`-based machines, respectively.

I. Introduction

Quantum computers attempt to leverage superposition states and entanglement between multiple quantum bits or *qubits* to efficiently solve problems that are intractable for classical computers. These operations between qubits form quantum *gates* that collectively form quantum *circuits*. However, current "noisy, intermediate scale quantum" (NISQ) machines face limited connectivity between qubits and significant reliability challenges from executing these quantum circuits. These challenges are from many sources of noise including energy decay, dephasing, and crosstalk among qubits [1]. Thus, the holistic co-design goal for the execution of algorithms on quantum machines is to minimize the depth of the circuit, since the depth of the circuit is directly related to the fidelity of the circuit through the cumulative effect of these internal and external sources of noise [2], [3].

Different quantum machines use different *basis gates* governed, in part, by the type of physical interactions inherent to the quantum hardware. In superconducting NISQ machines, IBM uses the cross-resonance basis gate, a gate that is locally equivalent to a `CNOT` [4]. This choice is convenient since many quantum algorithms are themselves written in the `CNOT` basis. Other basis gates include the Google `SYC` gate and the `iSWAP` gate, which is readily realizable by systems using Transmon qubits [5]–[7]. The number of basis operations required is determined by circuit decomposition. Moreover, restricted quantum topologies require the use of `SWAP` gates to move information between qubits. `SWAP` gates must also be decomposed into basis gates, and generally require the most basis gate operations to be realized.



(a) `CNOT` decomposes into two $\sqrt{\texttt{iSWAP}}$ gates

(b) `CNOT`+`SWAP` (`CNS`) also decomposes into 2 $\sqrt{\texttt{iSWAP}}$ gates

Fig. 1: Decomposition of `CNOT`, `CNS` into the `iSWAP` family

The process to decompose, place gates, and route (i.e., insert `SWAP` gates) on the physical machine is called transpilation. Minimizing circuit depth in the transpiler is NP-hard [8], thus, state-of-the-art techniques are largely heuristic, ranging from stochastic methods to subgraph isomorphism algorithms [9]. Due to the complexity, a customary abstraction barrier is inserted between routing and decomposition such that they are computed independently. This is designed to make the problems independent and tractable.

Unfortunately, this separation of concerns often disregards scenarios where a certain choice of where a `SWAP` is inserted can enhance quantum circuit compression, an advantage only detectable post basis-translation [10]. One significant example includes a two-qubit gate U immediately followed by a `SWAP` on the same circuit wires, which together form U', denoted as the *mirror gate* of U [11], [12].

A known mirror gate example is the relationship of CNOT and iSWAP, shown in Fig. 1. CNOT appears ubiquitously in quantum circuits due to its ability to create entangled states as well as defining subroutines including singular value transformations, stabilizer measurements, and the decomposition of multi-qubit gates [1]. Of course, as stated above, many superconducting qubits natively produce photon-exchange gates like iSWAP [6], [13], [14], or, in bosonic cases, 'beamsplitters' [15]. Moreover, recent work showed that CNOT could be decomposed into two $\sqrt{\texttt{iSWAP}}$ gates (Fig. 1a) [16]. However, the CNOT mirror gate, CNS or CNOT + SWAP, is locally equivalent to an iSWAP, i.e., two $\sqrt{\texttt{iSWAP}}$s (Fig. 1b) [17]. Thus, in the $\sqrt{\texttt{iSWAP}}$ basis, CNOT and CNS have the same circuit depth cost.

Thus, for iSWAP-based machines, the CNS provides *"free" data movement* using iSWAP versus the directly decomposing the regular CNOT gate using two $\sqrt{\texttt{iSWAP}}$ gates. However, the CNOT and iSWAP equivalency through CNS is only one, albeit useful, example that can be generalized. All gates have a mirror gate and both the standard and mirror gate can be decomposed into any basis gate. Given that SWAP gates are typically the most expensive gates to decompose in any basis gate, using a mirror gate that eliminates a SWAP, even a mirror with a higher decomposition cost, will result in reduced circuit depth. For instance, consider that a CNOT decomposition requires two SYC gates. If during transpilation, a SWAP between the outputs is necessary for routing, this will result in five SYC gates, two from CNOT and three from SWAP. However, the mirror of CNOT, locally equivalent to iSWAP, can be decomposed with three SYC gates, saving two gates.

Leveraging this property, we propose MIRAGE or *Mirror-decomposition Integrated Routing for Algorithm Gate Efficiency*. MIRAGE is a quantum co-design methodology based on utilizing mirror gates to aid in both decomposition and routing quantum circuits onto quantum machines. In MIRAGE we consider a *mirage* SWAP to be a SWAP gate that can be absorbed into another computational gate during decomposition, as in a CNS decomposing into iSWAP like in Fig. 1b.

Moreover, MIRAGE, unlike SABRE, breaks down the barrier between routing and decomposition. MIRAGE considers both the routing impact routing (adding SWAP gates) and the decomposition depth when selecting between a gate or its mirror. Like other heuristics, in MIRAGE starting conditions and condition(s) when to select mirror gates are highly impactful. Thus, we explore different *aggression levels*, which use different thresholds on when to insert mirror gates.

In particular, we propose the following contributions:

1) We demonstrate and quantify the value of using mirror gates for the iSWAP family of basis gates (including $\sqrt[n]{\texttt{iSWAP}}$ for $n = 2, 3, 4$ CNOT, and SYC basis gates.
2) We demonstrate mirror gates provide a similar decomposition benefit to approximate decomposition and that both approaches can be combined for additional reduction of infidelity and improved Haar scores.
3) We develop the MIRAGE transpilation flow that leverages mirror gates to benefit routing and decomposition.
4) We show when specifically optimizing for circuit depth rather than minimizing SWAP gates can achieve a significant improvement of 7.5%.
5) We conduct noise simulation and fidelity evaluation to demonstrate to impact of MIRAGE on iSWAP family,CNOT, and SYC basis gates.

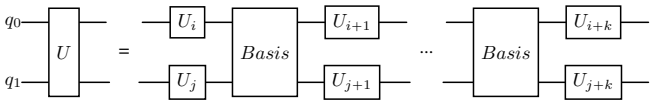


Fig. 2: Arbitrary basis gate decomposition template

Before we describe the details of our mirror gate analysis and the MIRAGE transpilation approach, we first provide some background in the next section.

II. Background

Quantum transpilers ① decompose gates in quantum algorithms onto the basis gate realizable in the target system, ② physically place the qubits and gates in the mapped algorithm to locations on the target machine, and ③ insert SWAP gates into the circuit to route states between qubits.

A common method to decompose an arbitrary two-qubit (2Q) gate, or *unitary* U in a basis gate uses Cartan's decomposition. In Cartan's KAK decomposition a 2Q U is factored by a series of 2Q basis gates combined with single-qubit unitaries(1Q) [18]. This method allows for the effective construction of any 2Q unitary operation as shown in Fig. 2 for a generalized form that requires k copies of the basis gate.

Placement and routing are important because quantum topologies are restricted. While many machines use a square lattice topology, as IBM machines scale, their routing flexibility eventually decreases to a heavy-hex lattice, which is currently standard [19]. This routing limitation is due to the inherent crosstalk between qubits of the IBM cross-resonance gate [20]. Thus, as topology flexibility reduces, the number of SWAP gates required to implement circuits increases.

However, deep circuits work against circuit fidelity because infidelity is quantitatively linked to the total execution time of the quantum circuit. Specifically, the overall cost in time is derived from the durations of individual 1Q and 2Q gates, along with the number of necessary k applications of the 2Q basis gate. As the duration, or depth, of a quantum circuit increases, the system becomes increasingly susceptible to noise, such as decoherence and dephasing, leading to a gradual degradation of the stored quantum information [21]. Additionally, the inclusion of SWAP operations contributes significantly to the overall duration and, consequently, the infidelity of executing a quantum circuit. Notably, for $\sqrt[n]{\texttt{iSWAP}}$ basis gates, SWAP operations demand the highest k values for decomposition, thereby becoming the dominant source of error [16], [22].

Various transpilation optimization techniques have been developed in recent years, such as finding gate commutation rules [23], pulse level decomposition optimizations [24] and eliminating gates for known pure-state inputs [25]. However,

The effectiveness of these solutions are highly context dependent, emphasizing the importance of comparative evaluations in diverse quantum computing scenarios. In the next section we describe the impact of using mirror gates in decomposition.

III. Mirror-Gate Decomposition

To understand the potential impact of decomposition using mirror gates requires a method for articulating how this process can contribute to circuit depth reduction. In this section we compute the advantage on computational power of different basis gates using mirror gates on unrestricted topologies (e.g., all-to-all networks), and then we explore how this maps to different restricted topologies. However, to define and understand certain terms, methodologies, and metrics, we start with some preliminaries discussed in the next section.

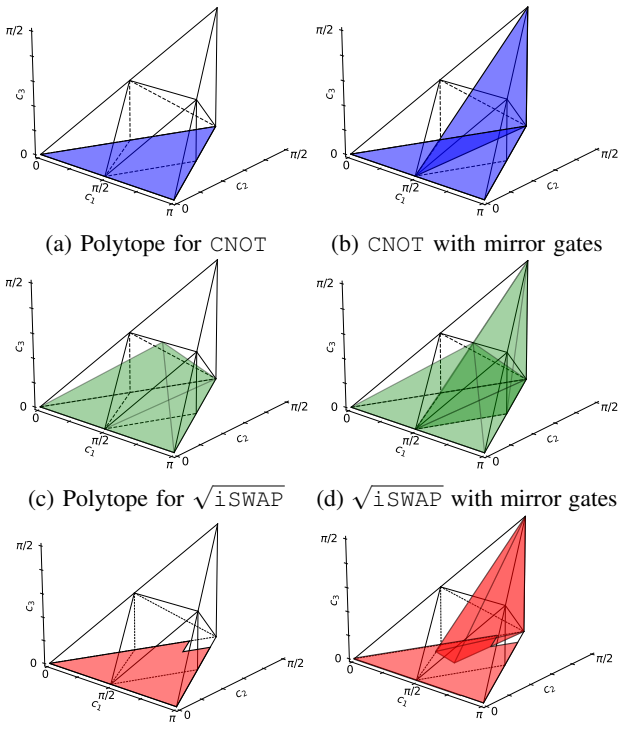
A. Preliminaries

In Section II we introduced the concept of decomposition into basis gates using Cartan's Decomposition. However, decomposition into sequences only the target hardware's basis gates has been a study of considerable research that can be divided into two categories: exact analytical decomposition and approximate numerical decomposition. For 2Q unitary targets, an optimal explicit set of rules has been derived for `CNOT` [26] and much more recently for the $\sqrt{\texttt{iSWAP}}$ [16], which is a half rotation $\pi/2$ (half pulse duration) of a full π rotation (full pulse) `iSWAP` [27].Generalizing decomposition to an arbitrary-sized unitary target is an open problem. The most efficient analytical method, Quantum Shannon Decomposition (QSD), is still far from optimal. For instance, QSD decompositions to `CNOT` gates are approximately twice as expensive as the theoretical lower bound [28]. Moreover, QSD and other methods such as Cosine-Sine Decomposition (CSD) only work for controlled-unitary basis gates [29], [30].

In contrast, numerical decomposition methods are more flexible than analytical decomposition and can closely approach the theoretical lower bounds of gate costs, even when scaling up to 5Q decompositions [28], [31], [32]. Numerical decomposition tunes the parameters of a *circuit ansatz* or starting guess for the form of the decomposed circuit. The better the form of the ansatz matches the optimal decomposition, the better quality of numerical solution.

As in Cartan's decomposition from Fig. 2, a reasonable numerical approach sequentially alternates applications of the 1Q and 2Q gates, to form different choices of the ansatz in an attempt to best match the form of an optimal or near-optimal solution. The suitability of the ansatz is determined through numerical optimization of the 1Q gates. The suitability of the solution can be measured using a similarity approach such as the Hilbert-Schmidt norm, which defines the distance between the ansatz and the current unitary [33]. However, the use of numerical methods becomes complex with an increase in the number of qubits, due to an exponentially expanding parameter space and the intricate generation of the ansatz.

To reason about decomposition, the requirements of a unitary from the quantum algorithm and the expressible regions



(a) Polytope for `CNOT` (b) `CNOT` with mirror gates

(c) Polytope for $\sqrt{\texttt{iSWAP}}$ (d) $\sqrt{\texttt{iSWAP}}$ with mirror gates

(e) Polytope for `SYC` (f) `SYC` with mirror gates

Fig. 3: Coverage comparison between standard and mirror-inclusive monodromy polytopes for the `CNOT` and $\sqrt{\texttt{iSWAP}}$ circuit ansatz with $k = 2$. Coverage under the standard scenario is juxtaposed with the coverage when mirror gates are allowed for mirror-inclusive polytopes.

within a given basis, a visual representation is *the Weyl chamber*. The Weyl chamber is a geometric framework derived from Lie Algebra that assigns a unique coordinate to each 2Q unitary operation, invariant under the application of 1Q gates [34]. As 2Q gates represent a unique point in the Weyl chamber, it is possible to describe regions of the Weyl chamber reachable through application of multiple 2Q gates using *monodromy polytopes* [35]. In particular, these polytopes define the accessible regions of a circuit ansatz for a given basis gate set. These regions, represented as convex polytopes, encapsulate the space of achievable 2Q unitaries within a fixed circuit depth. Examples of the Weyl chamber for the most common basis gates are shown in Fig. 3 with regions reachable by a circuit ansatz with $k = 2$ `CNOT` covering he class of offerings by IBM, Rigetti, IonQ, $\sqrt{\texttt{iSWAP}}$ representing the `iSWAP` family of gates naturally realized in superconducting systems, and `SYC` gates as deployed by Google are shown in Figs. 3a, 3c, and 3e, respectively.

B. Mirror Gates

As discussed in Section I, a *mirror gate* refers to the resulting gate formed by U composed with a `SWAP`. sThe `CNS` gate from Fig. 1 is a special case of mirror gate that has received considerable attention. It naturally appears in many common circuits. For instance, the Toffoli and Fredkin 3Q

gates can be decomposed using CNOT and CNS gates, allowing optimization at the gate-decomposition level [10], [17], [36]. The CNS gate also occurs naturally in stabilizer measurements of error-correcting codes [17], [37], [38], in entanglement purification protocols [39], and QAOA circuits [40]–[42]. In QFT circuits, the fractional controlled-phase gates can be replaced by CNS gates, allowing the QFT circuit to be implemented solely using iSWAP gates [43]. Moreover, the Diamond Gate, a native 4Q gate, can be recast into CNS gates, which can be used to build controlled-phase operations [44].

The transformation of an arbitrary U into its mirror U' has been described in the positive canonical basis, a conventional representation for the Weyl Chamber coordinates, in Eq. 1 [12], [45]. The two forms deal with the mirrored nature of the Weyl chamber at the midpoint in the line between CNOT and iSWAP.

$$(a', b', c') = \begin{cases} (\frac{\pi}{4} + c, \frac{\pi}{4} - b, \frac{\pi}{4} - a) & \text{if } a \leqslant \frac{\pi}{4} \\ (\frac{\pi}{4} - c, \frac{\pi}{4} - b, a - \frac{\pi}{4}) & \text{else} \end{cases} \tag{1}$$

In this work, we explore the use of monodromy polytopes that are extended to include mirror gates. Since polytopes denote the accessible regions within a fixed circuit depth, then if mirroring is permitted, the polytope should also encompass the mirror gates corresponding to every gate within the original region. Thus, we seek to identify the set of unitaries that are accessible, considering an allowance for permutations of output wires, e.g., unitaries that permit a *mirage* SWAP gate. This is particularly useful in scenarios where the order of the output wires is inconsequential, such as in richly connected topologies found in recently proposed superconducting qubit architectures with local All-to-All (A2A) connectivity [46].

To construct a mirror-permitted polytope, we assign a cost of zero to a SWAP operation, which serves to permute the order of output qubits. Then it is possible to create two polytopes that represent the portion of the Weyl Chamber reachable using a circuit ansatz with a particular value of k. For the CNOT, $\sqrt{\texttt{iSWAP}}$, and SYC gates, we depict the $k = 2$ case in Figs. 3b, 3d, and 3f[1] of the Weyl Chamber volume. In both the standard polytope and the mirror-permitted polytope for CNOT, the planar slices contribute to 0% volume coverage. In contrast, the $\sqrt{\texttt{iSWAP}}$ gate in its standard form covers 79.0% of the Haar-weighted volume and increases to 94.4% when mirror gates are utilized.

The mirror polytopes from Fig. 3 intersect. The CNOT intersection is a line from CNOT to iSWAP, whereas there is an appreciable region of overlap for $\sqrt{\texttt{iSWAP}}$. This is another way to illustrate the increased computational power of the $\sqrt{\texttt{iSWAP}}$ basis, but it also suggests that perhaps smaller pulse lengths of the iSWAP family of gates could be particularly useful as we take advantage of mirror gates as shown in Fig. 4. For instance, with $\sqrt[3]{\texttt{iSWAP}}$ with a $\pi/3$ or $1/3$ pulse iSWAP a substantial portion of the Weyl chamber is covered at $k = 2$

[1]The $k = 1$ case has 0% volume as these are only points in the chamber. The $k = 3$ case is less interesting for these gates as both the CNOT and $\sqrt{\texttt{iSWAP}}$ basis gates cover 100% and SYC covers >99%

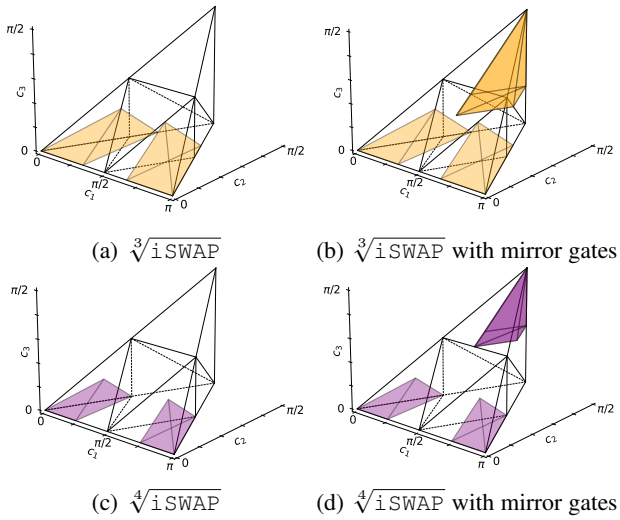


Fig. 4: Coverage comparison between standard and mirror-inclusive monodromy polytopes for the $\sqrt[3]{\texttt{iSWAP}}$ and $\sqrt[4]{\texttt{iSWAP}}$ circuit ansatz with $k = 2$.

(Fig. 4b). Even $\sqrt[4]{\texttt{iSWAP}}$ with a $\pi/4$ or $1/4$ pulse iSWAP has useful coverage (Fig. 4d) with the advantage that calibrating such a gate can easily form a $\sqrt{\texttt{iSWAP}}$ with two back-to-back pulses like two back-to-back $\sqrt{\texttt{iSWAP}}$ pulses form an iSWAP.

Interestingly, while CNOT is not reached, many of the CPHASE gates are in both cases, which is potentially useful in many algorithms. Furthermore, the maximum cost polytopes decrease with mirror gates for these partial iSWAPs; for example, using $\sqrt[4]{\texttt{iSWAP}}$ traditionally requires up to $k = 6$ depth (equivalent to $k = 3$ for $\sqrt{\texttt{iSWAP}}$), but with mirroring, the depth never exceeds $k = 4$, which guarantees full Weyl Chamber coverage at the equivalent of $k = 2$ $\sqrt{\texttt{iSWAP}}$s. In the next section we quantify this advantage and impact on decomposition fidelity.

C. Computing Fidelity

One mechanism to quantify the Weyl chamber coverage of a basis gate set is its Haar score, or Haar-average expected circuit cost. This computes the weighted average decomposition cost for a uniformly distributed random 2Q unitary [47]. This cost can be precisely computed using monodromy polytopes. While previous work has shown that mirror gates can enhance Haar scores, this has been done particularly in the context of super-controlled basis gates [12], [45]. Moreover, Qiskit's transpiler implementations have focused on the XX basis to simplify the computation [48].

Generalizing this concept to a broader set of operators requires a function that maps a point outside the polytope coverage region to the nearest point within that region. We use a numerical decomposition method to optimize an ansatz to the nearest point outside the coverage region in order to generalize this mirror gates methodology to arbitrary basis gates. Using this approach, we can examine the effects of mirror gates on the Haar scores for $\sqrt[n]{\texttt{iSWAP}}$ gates, among others.

Algorithm 1 Monte Carlo for approximate decomposition Haar scores

```
procedure ApproxGateCosts(N)
    TotalCost ← 0
    for i in range(N) do
        Target ← HaarSample()
        for each P in Set do
            if P contains Target then
                Compute ExactCost and FidThreshold
            end if
        end for
        BestCost ← ExactCost
        for each P in CheaperSet do
            Cost ← Optimize(P, Target, FidThreshold)
            if Cost ≠ None then
                BestCost ← min(BestCost, Cost)
            end if
        end for
        TotalCost ← TotalCost + BestCost
    end for
    return TotalCost/N
end procedure
```

Practical quantum computing can take advantage of approximate decompositions that may lead to higher overall fidelity because they require fewer applications of the noisy basis gate [31], [33], [46]. In other words the infidelity from the approximation is less than the infidelity from the increased noise resulting from the additional complexity of circuit necessary to compute the exact result. To measure this, we use an error model proposed in previous work [22], [49] that identifies decoherence over time as the main source of infidelity. This model defines the fidelity of a gate, F_Q, in terms of an exponential decay relative to the gate duration and the qubit's T_1 relaxation rate as described in Eq. 2 [50].

$$F_Q = e^{-\text{Gate Duration/Qubit Lifetime}} \tag{2}$$

Using this model, decomposition becomes an optimization problem that balances circuit and decomposition fidelity. The total fidelity, a product of these two fidelities, governs the acceptance threshold for a given circuit. This decomposition error tolerance threshold can be intuitively understood as expanding the volume of each of the coverage sets [51].

To compute updated Haar scores that factor in both mirror gates and approximate decompositions, we use Monte Carlo sampling of unitary targets from the Haar distribution, and verify decomposition fidelity using numerical decomposition. First, we calculate circuit infidelity from the exact decomposition solution before subsequently checking if any cheaper polytopes (corresponding to higher fidelity circuits) can approximate the target circuit within the prescribed total fidelity threshold. The overall process is encapsulated in Algorithm 1.

Presuming that iSWAP carries a normalized unit cost of 1.0 with fidelity of 99% [13] means fractional $\sqrt[n]{\texttt{iSWAP}}$ gates, which have shorter unit time costs have less decoherence time, e.g., $\sqrt{\texttt{iSWAP}}$ with 0.5 [27]) have proportionately adjusted fidelities. The final decomposition fidelity obtained through approximate decomposition is recorded for each iteration.

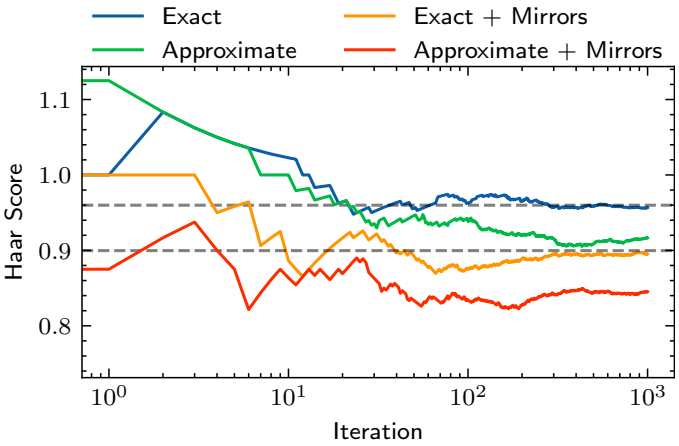


Fig. 5: Haar Score convergence for $\sqrt[4]{\texttt{iSWAP}}$ across 1000 iterations, for different optimization strategies. Horizontal lines denote exact values derived through polytope integration.

Basis Gate	Haar	Fidelity	Mirror Haar	Mirror Fidelity
$\sqrt[2]{\texttt{iSWAP}}$	1.105	0.9890	1.029	0.9897
$\sqrt[3]{\texttt{iSWAP}}$	0.9907	0.9901	0.9545	0.9904
$\sqrt[4]{\texttt{iSWAP}}$	0.9599	0.9904	0.8997	0.9910

TABLE I: Haar score and corresponding average fidelities for different basis gates, with and without mirror decomposition.

Basis Gate	Haar	Fidelity	Mirror Haar	Mirror Fidelity
$\sqrt[2]{\texttt{iSWAP}}$	1.031	0.9895	0.9950	0.9899
$\sqrt[3]{\texttt{iSWAP}}$	0.9433	0.9904	0.8900	0.9908
$\sqrt[4]{\texttt{iSWAP}}$	0.9165	0.9906	0.8453	0.9913

TABLE II: Haar score and corresponding average fidelities for different basis gates, including approximate decompositions.

The Monte Carlo convergence for $\sqrt[4]{\texttt{iSWAP}}$, subject to 1000 iterations, is illustrated in Fig. 5. It can be noted that the Exact and Exact + Mirrors solutions converge successfully at the theoretically computed values illustrated with dotted lines in the figure. This gives confidence that the decomposition that allows for approximate solutions also reasonably converges. Noting, a lower Haar score is more desirable, the approximate solution without mirrors nearly reaches the exact solution that contains mirror gates. By combining both, approximation and mirror gates, the Haar score improves from 0.9 to under 0.85.

The average total fidelities and Haar score for exact decomposition, both with and without allowed mirror gates, are recorded in Table I and are extended to allow for approximate decomposition when it improves fidelity in Table II.

The observed fidelity improvements are particularly noteworthy, given that they stem solely from compiler-level optimizations, without necessitating experimental adjustments (i.e., a hardware change). For example, **using $\sqrt{\texttt{iSWAP}}$ as the basis gate, the transition to approximate mirror decomposition provides an 8.8% relative decrease in total infidelity, and a 9.4% decrease when $\sqrt[4]{\texttt{iSWAP}}$ is used**. While smaller fractional basis gates can bolster Haar scores, previous work reveals that as these basis fractions diminish,

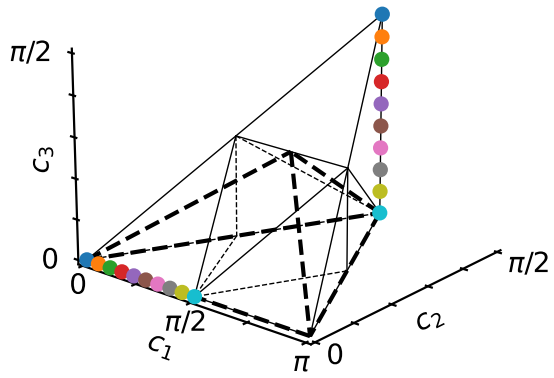


Fig. 6: CPHASE gates and their mirrored counterparts in pSWAP using the same color indicate corresponding gates. The black dashed lines denote coverage of $\sqrt{\texttt{iSWAP}}$ with $k = 2$. Notably, the light blue points, which represent CNOT and CNS, are the only gates from both groups contained by this region.

Basis Gate	Haar	CNOT	CNS	SWAP
$\sqrt{\texttt{iSWAP}}$	2.2	2	2	3
CNOT	3.0	1	2	3
SYC	3.0	2	3	3

TABLE III: Decomposition costs in terms of gate counts

the contribution of interleaving single-qubit gates becomes increasingly significant. Furthermore, the primary target of interest, CNOT, will not experience further improvements with progressively smaller fractional gates iSWAP, indicating a limit to this strategy [22]. Hence, this further supports $\sqrt{\texttt{iSWAP}}$ as a strong basis gate candidate.

D. Mirroring on Restricted Topologies

Although mirror gates will reduce gate decomposition costs in all-to-all topologies, in practical topologies with less rich connectivity, they may inadvertently add unfavorable qubit permutations, which would need to be undone using a SWAP, nullifying any benefit from the cheaper mirror decomposition. A transpilation algorithm is essential to find optimal use of mirror gates, taking into account subsequent gates to minimize both gate decomposition costs and SWAP operations.

Quantum computing algorithms frequently involve controlled-phase or CPHASE gates. Using Eq. 1, it is possible to mirror the CPHASE family into the parametric-SWAP family [52], depicted in Fig. 6 in the Weyl chamber. As seen in the figure, the $k = 2$ coverage region for a $\sqrt{\texttt{iSWAP}}$ basis is delineated by bold dashed lines. This region encompasses the CPHASE gates, but not all of the pSWAP gates. Intriguingly, both the CNOT gate and its mirrored counterpart (as shown in Fig. 1, such as CNS or iSWAP) reside within the $k = 2$ coverage area. Therefore, when mirroring, only for this pair does the decomposition cost remain constant.

Substituting in a gate's mirror to absorb required SWAP gates is not restricted solely to the CNOT gate paired with a iSWAP or $\sqrt{\texttt{iSWAP}}$ basis. For the CPHASE family, even if the pSWAP mirror has a increased decomposition cost ($k = 3$), it remain a favorable substitution if it eliminates a SWAP operation. As highlighted in Table III, mirroring a CNOT in the quantum algorithm does not always maintain the decomposition cost, especially when targeting other basis gates, including CNOT or SYC. While $\sqrt{\texttt{iSWAP}}$ is an exemplar for decomposition efficiency, the primary objective remains to eliminate SWAP operations, even if it introduces locally elevated decomposition costs.

In the next section, we will detail our routing algorithm, MIRAGE, which strategically uses mirror gates based on the broader context of the quantum circuit. The goal is not only to minimize costs by mirroring every possible gate, but to strategically place and select the decomposition of the mirrors considering downstream operations, topology constraints, and the potential to absorb SWAP operations.

IV. MIRAGE

In this section, we introduce MIRAGE, a collaboratively designed transpilation algorithm that collectively examines SWAP insertion and decomposition. MIRAGE can exploit the intrinsic efficiency of $\sqrt{\texttt{iSWAP}}$ gates, which decomposes both CNOT and CNS at the same cost, an advantage that is not available to the traditional CNOT basis (Fig. 1). However, conceptually, MIRAGE can be used for any basis gate. Using monodromy polytopes and mirror and approximation based decomposition discussed in Section III, MIRAGE can select between gates and their mirror form to determine the appropriate choice for implementation in the circuit.

MIRAGE is a heuristic approach with greedy characteristics similar to the prior state-of-the-art transpilation passes, such as SABRE [53]. However, unlike previous approaches, MIRAGE considers routing and decomposition cost when determining both SWAP placement and use of mirror gates. However, because MIRAGE remains fundamentally greedy, we consider several strategies that allow MIRAGE to avoid getting trapped into poorly performing situations, such as finding a local minimum far from the global optimum.

First, when considering a gate or SWAP choice, we evaluate the potential impact on circuit depth rather than solely SWAP depth as is considered in prior work. Second, we propose aggression levels which use different circuit depth thresholds to determine whether to insert a mirror gate. MIRAGE is implemented using the SABRE workflow included in Qiskit with several key modifications. We describe these next.

A. MIRAGE Workflow Details

The SABRE workflow provides several key data structures and a optimization flow that we inherit into MIRAGE. The circuit is represented as a directed acyclic graph (DAG). The gate nodes from the DAG are organized into a front layer, an execution layer, and a mapped layer. The mapped layer holds the portion of the DAG that has be mapped to the hardware.

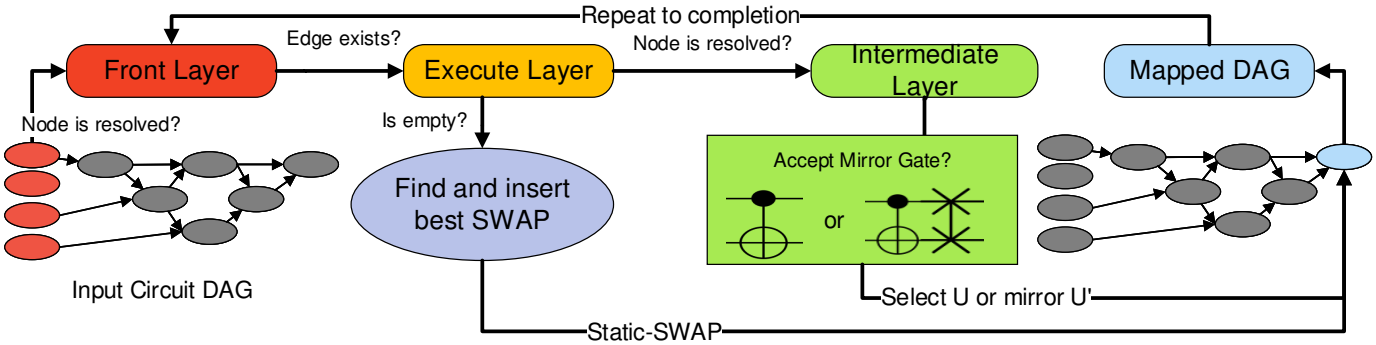


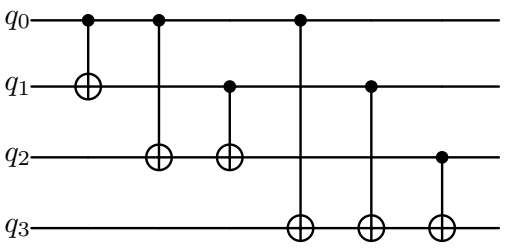
Fig. 7: MIRAGE Workflow

The front layer holds nodes with resolved dependencies, while the execution layer holds nodes that can be executed based on the current physical qubit layout. The fully processed nodes are transferred to the mapped layer into the mapped DAG.

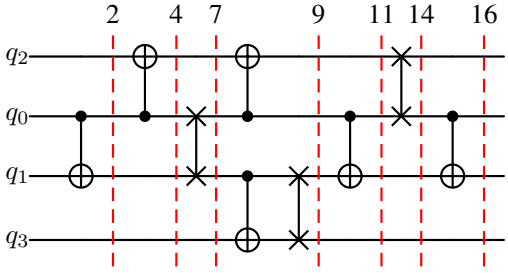
Gates move sequentially from the unmapped DAG to the front layer to the mapped layer, a process guided by the resolution of their predecessors. In other words, a gate can only progress to a stage once all its predecessor nodes have advanced past that stage. When the execution layer is empty, but unresolved gates remain in the front layer, SABRE selects some number of SWAP gates that are inserted into the execute layer with a goal of minimizing the heuristic cost function, designed to minimize topological distance between upcoming qubits and promote parallelism. This parallelism comes from selecting SWAP gates that can be executed at the same time to concurrently move data towards both inputs of the gate being mapped. Once sufficient SWAP gates are introduced, this stalled node in the front layer *advances*. After all gates in the front layer advance a complete mapping of all gates is accomplished in the mapped DAG. SABRE runs multiple times with different initial qubit mappings (seeds), which can dramatically change the results.

In MIRAGE we retain the front, execute, and mapped layers. However, we add an intermediate layer between the execute and mapped layer as shown in Fig. 7. The intermediate layer is used to handle every 2Q gate from the algorithm leaving the execution layer. The transpiler pass evaluates whether to accept the mirror gate in its place, then ultimately sends the chosen operation to the mapped layer. Note: in the original SABRE workflow, these 2Q gates are not decomposed to the basis gate of the target machine. However, in MIRAGE, while the gate is not decomposed, a decomposition estimate is considered in the intermediate layer and/or execution layer to improve the estimation of mirror or SWAP choices over pure topological distance and gate depth.

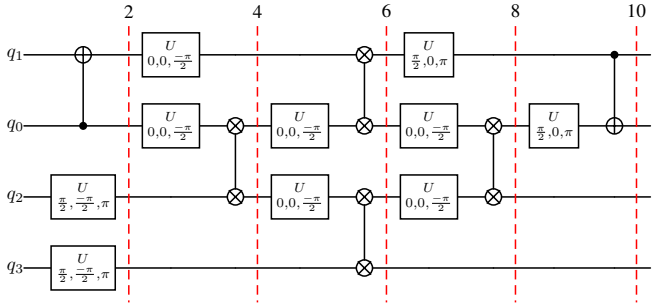
The impact of MIRAGE is illustrated in Fig. 8 targeting a $\sqrt{\texttt{iSWAP}}$ machine considering a fully entangling ansatz—a common structure in quantum machine learning and optimization algorithms (Fig. 8a). When this circuit is mapped to a line topology and optimized using Qiskit at level 3, the circuit depth amounts to 16 $\sqrt{\texttt{iSWAP}}$ basis gates using 3 SWAP gates (Fig. 8b). To interpret this pulse count, with $\sqrt{\texttt{iSWAP}}$ basis



(a) TwoLocal (full), 4 qubits



(b) TwoLocal (full), 4-qubit line topology, Qiskit level 3



(c) TwoLocal (full), 4-qubit line topology, MIRAGE

Fig. 8: Comparison of TwoLocal circuits

gates it requires $k = 2$ to implement a CNOT and $k = 3$ to implement a SWAP. Note, between pulses 7 and 9, the CNS gate between q_2 and q_3 can be realized as an iSWAP which is equivalent to $\sqrt{\texttt{iSWAP}}$ with $k = 2$.

In contrast, using MIRAGE, the SWAPs are better absorbed into gates. By inverting the first CNOT the next CNOT is between q_1 and q_2 eliminating a SWAP. Replacing that CNOT with a mirror sets up doing the third and fourth CNOT in

parallel because both gates are independent and both are one step away. Ultimately, MIRAGE finds the same circuit functionality with only 10 $\sqrt{\texttt{iSWAP}}$ gates and no SWAP gates, underscoring the potential optimization (Fig. 8c).

B. Post-Selection Metric

An important advancement of MIRAGE is the selection metric to choose the optimal routing among multiple independent trials. The original SABRE implementation relies on a decay factor to promote parallelism, discouraging the use of SWAP gates on qubits that have recently undergone a SWAP. However, when performing multiple routing trials, the tracked metric is the total number of induced gates SWAP, which is less closely related to the depth of the circuit [53]. This disconnect arises because the transpiler cannot reason about depth until it performs decomposition into the basis gate.

MIRAGE, addresses this by enabling the estimation of circuit depth without the need for actual decomposition. This is accomplished using monodromy polytopes to rapidly assess circuit costs. The minimum-cost circuit polytope that contains the unitary target is identified as discussed in Section III. We iterate the coverage set in until we find a coverage region containing the edge's 2Q gate. Thus, the depth metric is calculated using the longest DAG path with a custom weight function assigned to decomposition cost. Total gate counts are calculated similarly, summed over all nodes.

While exact methods for decomposing into CNOT and $\sqrt{\texttt{iSWAP}}$ exist, MIRAGE is designed to operate independently of any specific decomposition strategy. The actual decomposition can be specified later using either exact or numerical methods. While we continue to use decay when choosing individual SWAPs, the post-selection process is about choosing between routes across the independent trials.

C. Overcoming Local Minimas

Unfortunately, even with the best possible lookaheads, greedy algorithms have a propensity to get stuck in local minima that can be quite a bit worse than the optimal solution. We address the challenge of local minima that can obstruct the optimization process by noting that the algorithm's sensitivity to initial placement and cost decision making can lead to certain cases where the initial layout enters a non-converging cycle, as shown in Fig. 9.

In this example, a subset of the circuit from Fig. 8a, the qubits are reordered, so that the CNOT between q_0 and q_3, denoted as $\texttt{CNOT}_3^0$ requires no SWAPs. The best choice in the forward pass is to $\texttt{SWAP}_0^2$ on the top qubits while executing $\texttt{CNOT}_3^1$ using the bottom qubits. Finally, $\texttt{CNOT}_3^2$ executes, requiring a depth of seven pulses. However, the minimal solution requires two SWAP gates be added to further reorder the qubits. While one SWAP can be combined into a $\texttt{CNS}_3^0$ gate, because a $\texttt{SWAP}_0^1$ is necessary, the total pulses becomes nine. However, in the backwards pass with this new ordering, this leads to the solution on the bottom right of Fig. 9, such that the $\texttt{CNS}_3^1$ replaces the middle gate, creating a solution with only six pulses, while the local minima got stuck at seven.

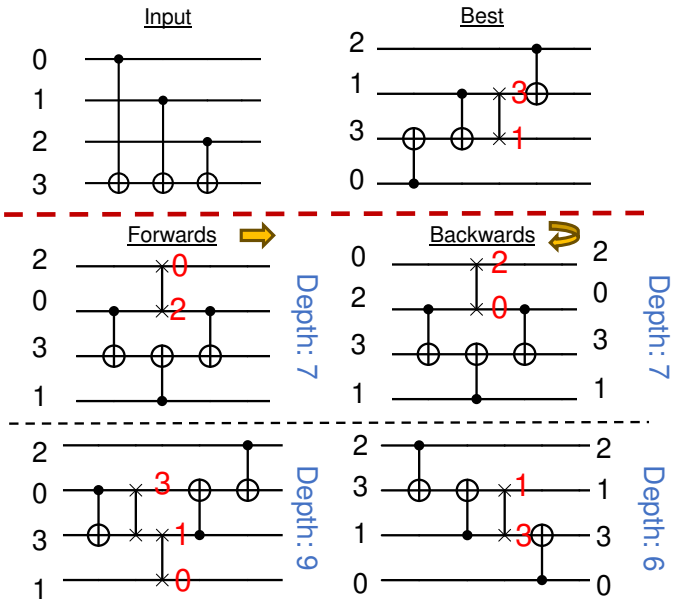


Fig. 9: Two routing trials from the same initial layout demonstrate the challenges of local decision-making. The top route, despite an initial optimal choice, ends in a local minima, while the bottom, through an initial sub-optimal choice, is the best solution.

To address this, we use a technique described as *mirror aggression settings* to dictate the likelihood of accepting a mirror gate decomposition in the intermediate layer. Specifically, we define four aggression levels: level 0, a mirror gate is never accepted; level 1, a mirror gate is accepted if it lowers the cost; level 2, a mirror gate is accepted if it either lowers or maintains the cost; and level 3, a mirror gate is always accepted.

In practice, different circuits perform well with different aggression levels. To evaluate the impact of fixed aggression settings, we conducted tests using each level of aggression (Fig. 10). We selected a subset of circuits to demonstrate that **no single strategy is universally optimal**. The results support the use of a mix of aggression settings, allowing MIRAGE to handle a wide range of circuits and topologies. Note, Mirage-a0 results in essentially the same configuration as Qiskit, but Fig. 10 shows slight advantage to Qiskit by having more seed passes compared to Mirage-a0. Based on these representative circuit trials, we distribute the routing trials (seeds) across aggressions as follows: 5% at level 0, 45% at level 1, 45% at level 2, and 5% at level 3. This means that all MIRAGE runs attempt mapping at each aggression level with effort (number of seeds) to each aggression level as dictated by the sensitivity study in Fig. 10. This approach, with fewer attempts on the edge cases (levels 0 and 3) and the majority of attempts (effort) on the metric-based choices improves the effectiveness of various circuits from different arbitrary initial layouts (seeds).

Algorithm 2 describes the final MIRAGE algorithm that combines variable aggression settings. This approach optimizes compression and reduces the number of layout trials

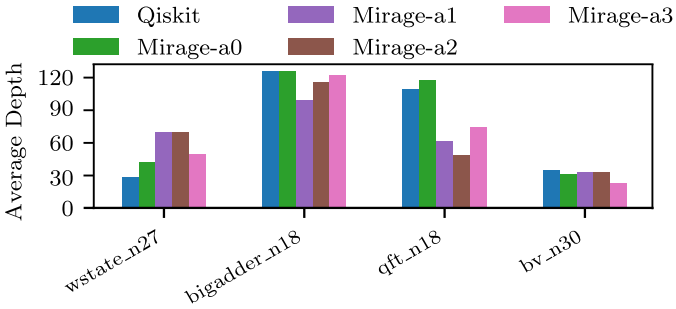


Fig. 10: Results of independent configuration trials with different aggression settings.

Algorithm 2 Mirror Gate Acceptance Function

```
Input: cost_current, cost_trial, aggression
Output: Boolean accept mirror gate
if aggression = 0 then
    return False
else if aggression = 1 and cost_trial <cost_current then
    return True
else if aggression = 2 and cost_trial ⩽ cost_current then
    return True
else if aggression = 3 then
    return True
end if
return False
```

required. Future work could explore further parameter tuning.

V. Experimental Setup

To evaluate the performance and effectiveness of MIRAGE we implemented the MIRAGE algorithm in `qiskit-terra` 0.24.2. To compute monodromy polytopes, we used the Qiskit-Extension tool, `monodromy` [35] which we adapted to compute cost-weighted integration for both the standard and mirror-permitted coverage polytopes.

Our setup does not exceed the parameters in the default SABRE workflow: a lookahead window size ($|E|$) of 20, a window weight (W_E) of 0.5, and a decay rate of 0.001, with a reset after every five steps or gate mapping. We addressed a limitation in Qiskit's SABRELayout, which omits independent layout trials when a custom routing pass, like MIRAGE, is specified. We modified SABRELayout to adhere to the original SABRE configuration when MIRAGE is specified: 20 independent layout trials, each with 4 forward and backward routing passes, each iteration routed independently 20 times.

To establish a baseline, we added the `CNOT` and `SWAP` decomposition rules to the session equivalence library, as Qiskit lacks support for $\sqrt{\texttt{iSWAP}}$. However, this was only done for final circuit output as our MIRAGE cost functions consolidate all blocks before calculating costs using `monodromy`. Moreover, for decomposition to an arbitrary basis, `monodromy` identifies the minimum cost polytope that contains a target gate in its region. The resulting circuit ansatz formed from the basis gates is then optimized by fitting the 1Q gate parameters using a numerical optimizer.

When executing the transpiler, the pass manager conducts input cleaning, which includes unrolling gates with more than two qubits, removing `SWAP`s, barriers, and identity gates. We consolidate all consecutive unitary blocks, ensuring MIRAGE operates solely on 2Q gates. We then check if an implementation with no `SWAP` gates can be found using VF2Layout. We invoke MIRAGE (or SABRE) if no `SWAP` gate free placement can be found. Subsequently, we incorporate Qiskit's remaining optimizations and reconsolidate the circuit.

Lastly, for speed comparisons, we compared MIRAGE to the most recent version of SABRE in Python. While a version of SABRE has been ported to Rust, this made it impractical to determine the impact of MIRAGE on runtime.

We evaluate MIRAGE's performance using circuits from QASMBench [54] and MQTBench [55]. These benchmarks are selected for their relevance to NISQ devices and their need for > 0 `SWAP` gates. This is crucial because our transpiler, like the stock Qiskit implementation, checks using VF2Layout if an optimal mapping exists that requires no `SWAP` gates. Consequently, for circuits like GHZ or any linear ansatz VQA, both transpilers would behave identically, and neither SABRE nor MIRAGE would be invoked.

Name	Qubits	2Q Gates	Class
wstate [54]	27	52	Entanglement
qftentangled [55]	16	279	Hidden Subgroup
qpeexact [55]	16	261	Hidden Subgroup
ae [55]	16	240	Hidden Subgroup
qft [55]	18	306	Hidden Subgroup
bv [54]	30	18	Hidden Subgroup
multiplier [54]	15	246	Arithmetic
bigadder [54]	18	130	Arithmetic
qec9xz [54]	17	32	EC
seca [54]	11	84	EC
qram [54]	20	92	Memory
sat [54]	11	252	QML
portfolioqaoa [55]	16	720	QML
knn [54]	25	96	QML
swap_test [54]	25	96	QML

TABLE IV: Selected circuit benchmarks

We measure transpiler success using circuit costs via normalized critical path durations. In our convention, an `iSWAP` gate has a time cost of 1.0, and a $\sqrt{\texttt{iSWAP}}$ has 0.5, as defined in Section III. We report the geometric mean of circuit depth across 5 instances for each experiment. Our focus on reducing circuit depth, compared to Qiskit SABRE, underscores MIRAGE's utility as a straightforward yet significant enhancement to the existing Qiskit routing stage.

VI. Results

In this section we examine the efficiency of MIRAGE. First we explore the impact of circuit depth comparison against number of gates, then we examine the effectiveness of the MIRAGE optimization for different common quantum machine topologies, and finally we consider the runtime impact of MIRAGE against prior work. Note, all experiments use the aggression level balancing discussed in Section IV-C.

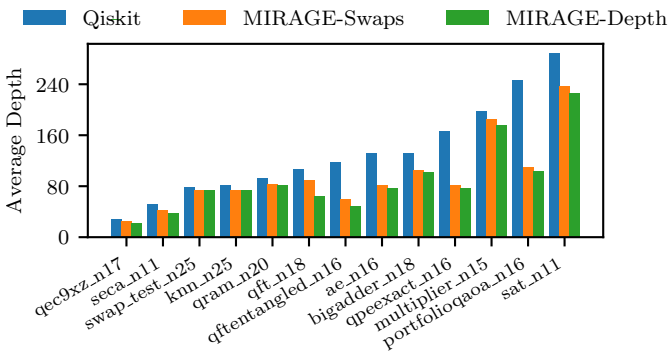


Fig. 11: Average circuit depth comparison for Qiskit, Mirage-SWAPs, and Mirage-Depth post-selection strategies.

A. Impact of Depth Comparison Metric

Previous transpilers tracked the circuit depth in terms of number of SWAP gates added to the circuit. MIRAGE improves on this by using a circuit depth metric to determine the best result as discussed in Section IV-B. In Fig. 11 we compared to the stock Qiskit SABRE, which determines the best result in of the fewest SWAP gates added, with MIRAGE using the same metric as well as our circuit depth cost function that more efficiently promotes parallelism. Our results indicate that both methods using MIRAGE find an improvement, due to inclusion of mirror gates. However, when we optimize for minimum swaps, we find an average depth reduction of 24.1%, and when we optimize for depth, we get **an additional 7.5%** relative benefit, leading to a total average depth reduction of 29.5%. Interestingly, the total number of gates is mostly unchanged (an increase of 0.4%), indicating that changing the selection metric is responsible for finding more parallelism.

B. MIRAGE for Common Quantum Machine Topologies

We evaluated MIRAGE using $\sqrt{\texttt{iSWAP}}$ gates targeting two production quantum machine topologies: 57Q Heavy-Hex and 6x6 Square-Lattice, tracking: critical path depthand number of SWAP gates. We report both an average and circuit size weighted average improvement. The results of MIRAGE on quantum circuit routing are shown in Fig. 12.

For Heavy-Hex, we observed an average and weighted average decrease of 31.2% and 33.4%, respectively, in circuit depth (Fig. 12a). The largest decrease in depth was 'qpeexact_n16' with a mirror gate acceptance rate of 95%, while the least was 'knn_n25' with a mirror gate acceptance rate of only 43%. **The average decrease of SWAPs was 56.2%, or a weighted average decrease of 75.8%.** The greatest decrease in SWAP gates was 'portfolioqaoa_n16' with a mirror gate acceptance rate of 99%, while the worst was'qram_n20'. **For the Square-Lattice topology, we observed an average and weighted average decrease of 30.0% and 32.1%, respectively in circuit depth** (Fig.12b). The largest decrease in depth was 'portfolioqaoa_n16' with a mirror gate acceptance rate of 100%, while the least was 'swap_test_n25' with a mirror gate acceptance rate of only 34%. **The average decrease of SWAP gates was 59.9%, or a weighted average decrease of 77.6%**. The greatest decrease in SWAP gates was 'seca_n11' with a mirror gate acceptance rate of 28%, while the worst was again 'qram_n20'.

	Qiskit		MIRAGE	
Circuit	**Duration** ns	**Fidelity**	**Duration** ns	**Fidelity**
ae_n8 [55]	5800	0.274	3925	0.362
dj_n8 [55]	1950	0.756	1425	0.806
fredkin_n3 [54]	1500	0.797	1000	0.855
qft_n4 [55]	1300	0.813	1000	0.842
qftentangled_n8 [55]	7975	0.107	7300	0.119
qpeexact_n8 [55]	7600	0.208	6925	0.200
toffoli_n3 [54]	1000	0.871	750	0.894

TABLE V: Circuit depth and noise simulation fidelity

C. MIRAGE for CNOT and SYC Basis Gates

We assessed MIRAGE for the CNOT gate on a Heavy-Hex topology (typical for IBM machines) and the SYC gate on a Square-Lattice topology (used by Google). **Our results, in Fig.13, show that MIRAGE still provides significant benefits with the CNOT Heavy-Hex architecture seeing depth reduction of 25.5% and SYC Square-Lattice achieving 23.6% reduction.** A few benchmarks with CNOT and SYC showed minor degredations using MIRAGE, e.g., multiplier_n15 on the CNOT Heavy-Hex architecture. This is due to differing decomposition costs for mirror gates (Section III-D). The cost to decompose CNS using CNOT or SYC means each mirror must offset the costs of SWAP gates which MIRAGE cannot guarantee. Nonetheless, the overall advantage is substantial.

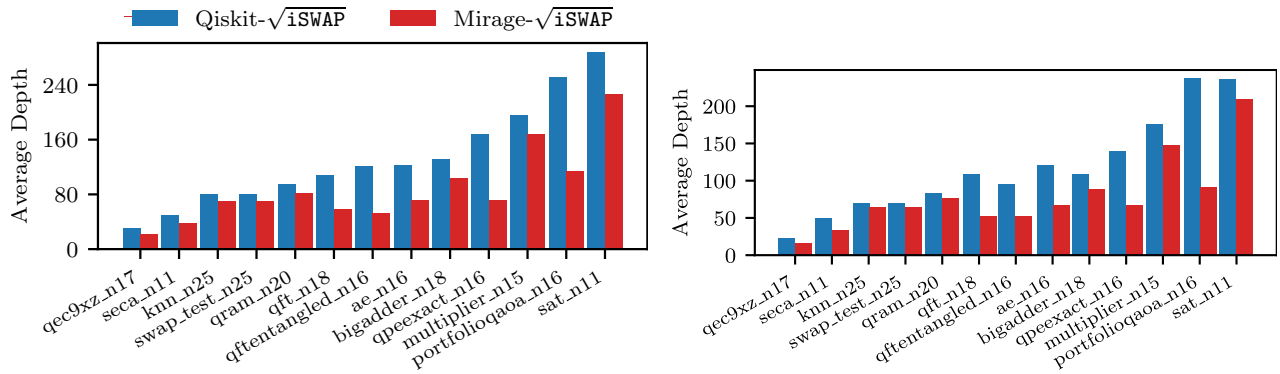
D. Noisy Fidelity Simulation

To quantify circuit fidelity we calculate the final circuit state fidelity F_T by comparing the noiseless simulation state ρ_1 with the noisy simulation state ρ_2 as described in Eq. 3. For noisy simulations, we utilized the Qiskit Aer simulator, incorporating both thermal relaxation and depolarizing error channels. Specifically, we set $T_1 = T_2 = 80\,\mu s$, with a single-qubit gate time of $25\,ns$ and a two-qubit iSWAP gate time of $100\,ns$ based on calculations from the SNAIL experimental prototype [13], [56]. We show results in Table V for smaller circuits implemented with Qiskit and MIRAGE on an 8-qubit machine with a line topology. The fidelity improvements are evident and correlate to the circuit depth improvements.

$$F(\rho_1,\rho_2) = \mathrm{Tr}[\sqrt{\sqrt{\rho_1}\rho_2\sqrt{\rho_1}}]^2 \quad (3) \qquad F_T = e^{-ND_C\left(\frac{1}{T_1}+\frac{1}{T_2}\right)} \quad (4)$$

To scale to larger circuits, we estimate fidelity using Eq. 4, based on the same principle from Eq. 2, but expanded to consider T_1 and T_2 for *all* N qubits for the duration of the circuit D_C [49]. Using the same circuit parameters for 1Q, and 2Q gate times; however, we choose near state-of-the-art T_1 and T_2 values scaled by a factor of 20 to work with more reasonable fidelity outcomes for the larger circuits.

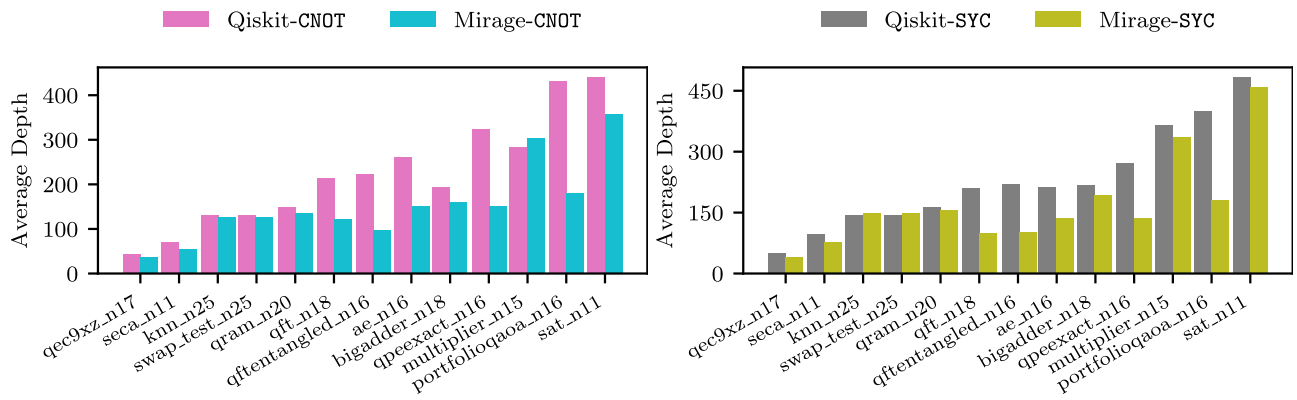
Our results, depicted in Fig. 14, highlight the average relative change in infidelity for different configurations. MIRAGE with $\sqrt{\texttt{iSWAP}}$ on Heavy-Hex shows a pronounced average decrease in relative infidelity of approximately 28.0%. In contrast, CNOT on Heavy-Hex achieves an average decrease



(a) Critical path depth for Heavy-Hex topology

(b) Critical path depth for Square-Lattice topology

Fig. 12: Optimization results with $\sqrt{\texttt{iSWAP}}$ basis gates, comparing between Qiskit-SABRE and MIRAGE.



(a) Critical path depth for Heavy-Hex topology

(b) Critical path depth for Square-Lattice topology

Fig. 13: Optimization results with CNOT and SYC basis gates,comparing between Qiskit-SABRE and MIRAGE.

in relative infidelity of about 21.1%, while SYC on a lattice topology records an average reduction of roughly 19.7%. The pronounced benefit observed for $\sqrt{\texttt{iSWAP}}$ underscores the efficacy of our transpilation optimization for this particular basis gate. As above, if the SWAP is not eliminated in the routing, the decomposition cost can increase making the optimization more of a possible tradeoff than in the CNOT vs. CNS case for $\sqrt{\texttt{iSWAP}}$. The topology certainly can play a role in the fidelity benefits as seen in the larger improvement for fidelity on Heavy-Hex than Square-Lattice for $\sqrt{\texttt{iSWAP}}$, however, the circuit and target basis gate also have a key role, noting that SYC on Square-Lattice, even with some negative outliers, out improves (on average) $\sqrt{\texttt{iSWAP}}$ on Heavy-Hex.

E. Implementation Improvements: Caching and Parallelism

To ensure a reasonable runtime of MIRAGE we profiled the tool and address the computational bottlenecks associated with the intermediate representation of the DAG nodes using coordinates. Profiling revealed that the conversion of a unitary to a coordinate and the UnitaryGate constructor were among the most expensive operations. To mitigate these issues, we introduced caching and parallelism into our approach.

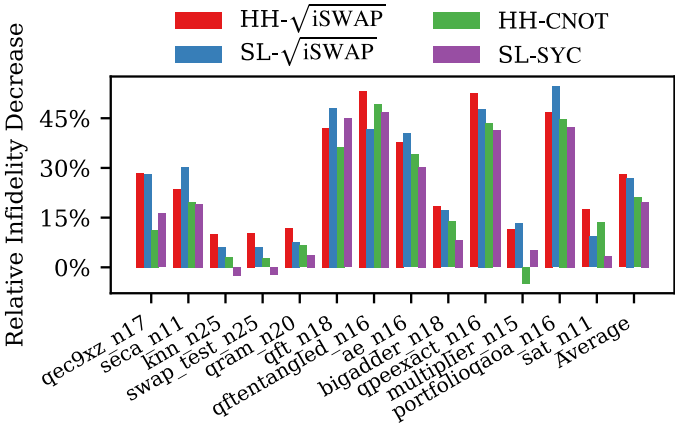


Fig. 14: Comparison of fidelity improvements from MIRAGE against Qiskit, for different basis gates and topologies.

In the intermediate layer, when a mirror gate is accepted, we replace the entire DAG node with a new unitary, rather than appending a gate that would disrupt consolidation. To efficiently build this unitary, we removed the costly calls to is_unitary and is_identity in the UnitaryGate

constructor, as we know that mirroring will always preserve unitarity. Finally, instead of computing the new mirror gate's coordinate, we use an equation to convert the original coordinate to the mirror coordinate (Eq. 1). Every SWAP gate added by MIRAGE comes with the SWAP coordinate manually annotated to the node.

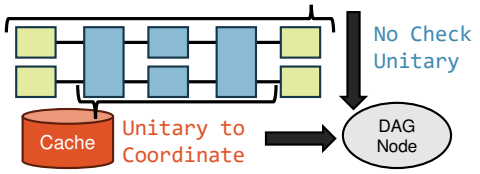
To further speed up conversion from unitary to coordinates, we rewrote the ConsolidateBlocksocks pass. Rather than multiplying all the operators into a new unitary, we first skip the exterior 1Q gates (since these will not change the coordinate), use that unitary as the cache key, annotate the block, and then multiply in the remaining gates (Fig. 15). This approach increases cache hits by removing the 1Q unitaries that would otherwise lead to different unitaries and ensures that each unitary has an annotated coordinate, eliminating the need for repeated calculations.

Additionally, we introduced a Least Recently Used (LRU) software cache, acting as a lookup table for each circuit polytope. This significantly diminishes the time required to query the same coordinates multiple times. Through this cache, each coordinate is ensured to be queried only once, thereby slashing the count of resource-intensive iterations over polytopes and calls to the has_element function. In comparison, our method outperforms Qiskit's Python transpiler notably. **For instance, when tested on a 64Q QFT circuit, MIRAGE executed 47.9% faster than Qiskit**. The code's speed directly impacts the number of individual trials, which subsequently influences the transpiler's solution quality. Therefore, MIRAGE can effectively optimize circuit depth across various circuit topologies by merely leveraging mirror gates efficiently, all without increasing the transpilation runtime.

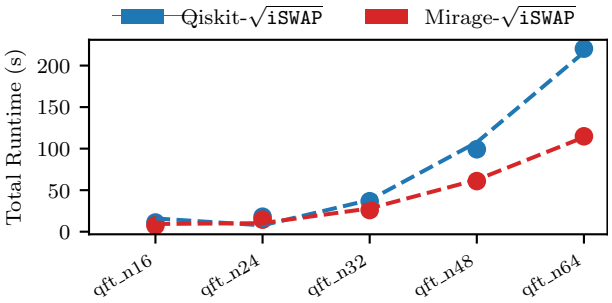
VII. Related Work

Additionally, several modifications have been proposed to further improve the effectiveness of SABRE. The most relevent are "Not all Swaps have the Same Cost" (NASSC), which refines the SABRE SWAP selection heuristic to select the most efficient SWAP by additionally considering potential CNOT gate cancellations due to encoded commutation rules [57]. The transpiler analyzes the set of adjacent nodes for each SWAP candidate to identify potential cancellations. For example, it considers a SWAP to be three inverted CNOTs and it looks for cases where those CNOTs can be commuted in the circuit such that the first and last CNOT from two SWAP gates can be cancelled. Like MIRAGE, NASSC does consider decomposition, but it is a very specific optimization to CNOT, while MIRAGE is much more general to the basis gate.

Permutation-aware Synthesis and Permutation-aware Mapping (PAS+PAM) searches for alternative decompositions of unitary blocks before committing any particular node to the mapped DAG [58]. PAS+PAM considers the potential for synthesizing an equivalent operation with permuted input and output qubits, which may reduce routing overhead. This is similar to MIRAGE. However, PAS+PAM, like NASSC, focuses on CNOT basis gates whereas MIRAGE is general to any basis gate. While PAS+PAM attempts to fully link



(a) Caching mechanism for building unitary Weyl coordinates



(b) Measured runtimes of Quantum Fourier Transform

Fig. 15: MIRAGE performance enhancements

decomposition and routing, MIRAGE uses decomposition information determined through monodromy polytopes, again which is agnostic to the type of basis gate, to track circuit depth and promote gate combination while routing to maintain an efficient runtime rather than fully combining the steps. Moreover, MIRAGE reports >30% reduction in circuit depth versus 18% reported by PAS+PAM to the same Qiskit baseline.

VIII. Conclusion

In this work, we have proposed MIRAGE, a strategy that integrates routing and decomposition in the transpiler, breaking the traditional abstraction of routing and decomposition. Our experiments show tangible improvements: For the Heavy-Hex topology, there was an average reduction of 31.19% in circuit depth, 16.97% decrease in total gate count, accomplished by a noteworthy 56.19% decrease of SWAPs. Meanwhile, the Square-Lattice topology experienced an average decrease of 29.58% in circuit depth and a 59.86% reduction in SWAP gates resulting in a reduction of infidelity of 28%. Applying MIRAGE to CNOT and SYC basis gates demonstrated circuit depth reduction of 25.5% and 23.6%, respectively, resulting in an infidelity reduction of 21.1% and 19.7%, respectively.

Future work aims to integrate MIRAGE directly into the transpiler pipeline using plugins. This allows us to select the routing trial that minimizes the estimated circuit depth, making each iteration of our program more geared towards depth reduction. Lastly, finding approximate decompositions without Monte Carlo methods could be done using quadratic programming or affine subspace projections to accelerate our approach for other basis gates.

IX. Software Availability

MIRAGE is open source and available online through github and will be provided for artifact evaluation.

ACKNOWLEDGEMENTS

This work is partially supported by The Charles E. Kaufman Foundation of The Pittsburgh Foundation under New Initiative Award KA2022-129519 and the University of Pittsburgh via a SEEDER grant.

REFERENCES

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

[2] J. Emerson, R. Alicki, and K. Życzkowski, "Scalable noise estimation with random unitary operators," *Journal of Optics B: Quantum and Semiclassical Optics*, vol. 7, no. 10, p. S347, 2005.

[3] Y. Y. Gao, M. A. Rol, S. Touzard, and C. Wang, "Practical guide for building superconducting quantum devices," *PRX Quantum*, vol. 2, no. 4, p. 040202, 2021.

[4] C. Rigetti and M. Devoret, "Fully microwave-tunable universal gates in superconducting qubits with linear couplings and fixed transition frequencies," *Physical Review B*, vol. 81, no. 13, p. 134507, 2010.

[5] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[6] Y. Sung, L. Ding, J. Braumüller, A. Vepsäläinen, B. Kannan, M. Kjaergaard, A. Greene, G. O. Samach, C. McNally, D. Kim *et al.*, "Realization of high-fidelity cz and z z-free iswap gates with a tunable coupler," *Physical Review X*, vol. 11, no. 2, p. 021058, 2021.

[7] M. Roth, M. Ganzhorn, N. Moll, S. Filipp, G. Salis, and S. Schmidt, "Analysis of a parametrically driven exchange-type gate and a two-photon excitation gate between superconducting qubits," *Physical Review A*, vol. 96, no. 6, p. 062323, 2017.

[8] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, "On the qubit routing problem," *arXiv preprint arXiv:1902.08091*, 2019.

[9] G. Nannicini, L. S. Bishop, O. Günlük, and P. Jurcevic, "Optimal qubit assignment and routing via integer programming," *ACM Transactions on Quantum Computing*, vol. 4, no. 1, pp. 1–31, 2022.

[10] J. Liu, M. Bowman, P. Gokhale, S. Dangwal, J. Larson, F. T. Chong, and P. D. Hovland, "Qcontext: Context-aware decomposition for quantum gates," *arXiv preprint arXiv:2302.02003*, 2023.

[11] P. Jurcevic, A. Javadi-Abhari, L. S. Bishop, I. Lauer, D. F. Bogorin, M. Brink, L. Capelluto, O. Günlük, T. Itoko, N. Kanazawa *et al.*, "Demonstration of quantum volume 64 on a superconducting quantum computing system," *Quantum Science and Technology*, vol. 6, no. 2, p. 025020, 2021.

[12] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, "Validating quantum computers using randomized model circuits," *Physical Review A*, vol. 100, no. 3, p. 032328, 2019.

[13] C. Zhou, P. Lu, M. Praquin, T.-C. Chien, R. Kaufman, X. Cao, M. Xia, R. S. Mong, W. Pfaff, D. Pekker *et al.*, "Realizing all-to-all couplings among detachable quantum modules using a microwave quantum state router," *npj Quantum Information*, vol. 9, no. 1, p. 54, 2023.

[14] P. Mundada, G. Zhang, T. Hazard, and A. Houck, "Suppression of qubit crosstalk in a tunable coupling superconducting circuit," *Physical Review Applied*, vol. 12, no. 5, p. 054023, 2019.

[15] Y. Lu, A. Maiti, J. W. Garmon, S. Ganjam, Y. Zhang, J. Claes, L. Frunzio, S. Girvin, and R. J. Schoelkopf, "A high-fidelity microwave beamsplitter with a parity-protected converter," *arXiv preprint arXiv:2303.00959*, 2023.

[16] C. Huang, T. Wang, F. Wu, D. Ding, Q. Ye, L. Kong, F. Zhang, X. Ni, Z. Song, Y. Shi *et al.*, "Quantum instruction set design for performance," *Physical Review Letters*, vol. 130, no. 7, p. 070601, 2023.

[17] N. Schuch and J. Siewert, "Natural two-qubit gate for quantum computation using the xy interaction," *Physical Review A*, vol. 67, no. 3, p. 032301, 2003.

[18] R. R. Tucci, "An introduction to cartan's kak decomposition for qc programmers," *arXiv preprint quant-ph/0507171*, 2005.

[19] J. M. Chow, A. D. Córcoles, J. M. Gambetta, C. Rigetti, B. R. Johnson, J. A. Smolin, J. R. Rozen, G. A. Keefe, M. B. Rothwell, M. B. Ketchen, and M. Steffen, "Simple all-microwave entangling gate for fixed-frequency superconducting qubits," *Physical review letters*, vol. 107, no. 8, p. 080502, 2011.

[20] F. Yan, P. Krantz, Y. Sung, M. Kjaergaard, D. L. Campbell, T. P. Orlando, S. Gustavsson, and W. D. Oliver, "Tunable coupling scheme for implementing high-fidelity two-qubit gates," *Physical Review Applied*, vol. 10, no. 5, p. 054062, 2018.

[21] W. G. Unruh, "Maintaining coherence in quantum computers," *Physical Review A*, vol. 51, no. 2, p. 992, 1995.

[22] E. McKinney, C. Zhou, M. Xia, M. Hatridge, and A. K. Jones, "Parallel driving for fast quantum computing under speed limits," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[23] M. A. Bowman, P. Gokhale, J. Larson, J. Liu, and M. Suchara, "Hardware-conscious optimization of the quantum toffoli gate," *arXiv preprint arXiv:2209.02669*, 2022.

[24] N. Earnest, C. Tornow, and D. J. Egger, "Pulse-efficient circuit transpilation for quantum applications on cross-resonance-based hardware," *Physical Review Research*, vol. 3, no. 4, p. 043088, 2021.

[25] J. Liu, L. Bello, and H. Zhou, "Relaxed peephole optimization: A novel compiler optimization for quantum circuits," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2021, pp. 301–314.

[26] F. Vatan and C. Williams, "Optimal quantum circuits for general two-qubit gates," *Physical Review A*, vol. 69, no. 3, p. 032315, 2004.

[27] D. R. Pérez, P. Varosy, Z. Li, T. Roy, E. Kapit, and D. Schuster, "Error-divisible two-qubit gates," *Phys. Rev. Appl.*, vol. 19, p. 024043, Feb 2023. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevApplied.19.024043

[28] P. Rakyta and Z. Zimborás, "Approaching the theoretical limit in quantum gate decomposition," *Quantum*, vol. 6, p. 710, 2022.

[29] M. Saeedi, M. Arabzadeh, M. S. Zamani, and M. Sedighi, "Block-based quantum-logic synthesis," *arXiv preprint arXiv:1011.2159*, 2010.

[30] V. V. Shende, S. S. Bullock, and I. L. Markov, "Synthesis of quantum logic circuits," in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, 2005, pp. 272–275.

[31] P. Rakyta and Z. Zimborás, "Efficient quantum gate decomposition via adaptive circuit compression," *arXiv preprint arXiv:2203.04426*, 2022.

[32] E. Younis, C. C. Iancu, W. Lavrijsen, M. Davis, E. Smith, and USDOE, "Berkeley quantum synthesis toolkit (bqskit) v1," 4 2021. [Online]. Available: https://www.osti.gov//servlets/purl/1785933

[33] L. Lao, P. Murali, M. Martonosi, and D. Browne, "Designing calibration and expressivity-efficient instruction sets for quantum computing," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 846–859.

[34] J. Zhang, J. Vala, S. Sastry, and K. B. Whaley, "Geometric theory of nonlocal two-qubit operations," *Physical Review A*, vol. 67, no. 4, p. 042313, 2003.

[35] E. C. Peterson, "`monodromy`: Computations in the monodromy polytope for quantum gate sets," 2021, https://github.com/Qiskit/monodromy.

[36] P. M. Cruz and B. Murta, "Shallow unitary decompositions of quantum fredkin and toffoli gates for connectivity-aware equivalent circuit averaging," *arXiv preprint arXiv:2305.18128*, 2023.

[37] A. Antipov, E. Kiktenko, and A. Fedorov, "Realizing a class of stabilizer quantum error correction codes using a single ancilla and circular connectivity," *Physical Review A*, vol. 107, no. 3, p. 032403, 2023.

[38] I. Simakov, I. Besedin *et al.*, "Scalable quantum error correction code on a ring topology of qubits," *arXiv preprint arXiv:2211.03094*, 2022.

[39] T. Tanamoto, K. Maruyama, Y.-x. Liu, X. Hu, and F. Nori, "Efficient purification protocols using iswap gates in solid-state qubits," *Physical Review A*, vol. 78, no. 6, p. 062313, 2008.

[40] Y. Ji, K. F. Koenig, and I. Polian, "Optimizing qaoa on bipotent architectures," *arXiv preprint arXiv:2303.13109*, 2023.

[41] B. Tan and J. Cong, "Optimal qubit mapping with simultaneous gate absorption," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–8.

[42] A. Hashim, R. Rines, V. Omole, R. K. Naik, J. M. Kreikebaum, D. I. Santiago, F. T. Chong, I. Siddiqi, and P. Gokhale, "Optimized fermionic swap networks with equivalent circuit averaging for qaoa," *arXiv preprint arXiv:2111.04572*, 2021.

[43] H.-F. Wang, X.-X. Jiang, S. Zhang, and K.-H. Yeon, "Efficient quantum circuit for implementing discrete quantum fourier transform in solid-state qubits," *Journal of Physics B: Atomic, Molecular and Optical Physics*, vol. 44, no. 11, p. 115502, 2011.

[44] E. Bahnsen, S. Rasmussen, N. Loft, and N. Zinner, "Application of the diamond gate in quantum fourier transformations and quantum machine learning," *Physical Review Applied*, vol. 17, no. 2, p. 024053, 2022.
[45] E. C. Peterson, L. S. Bishop, and A. Javadi-Abhari, "Optimal synthesis into fixed xx interactions," *Quantum*, vol. 6, p. 696, 2022.
[46] E. McKinney, M. Xia, C. Zhou, P. Lu, M. Hatridge, and A. K. Jones, "Co-designed architectures for modular superconducting quantum computers," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 759–772.
[47] K. Zyczkowski and M. Kus, "Random unitary matrices," *Journal of Physics A: Mathematical and General*, vol. 27, no. 12, p. 4235, 1994.
[48] Qiskit contributors, "Qiskit: An open-source framework for quantum computing," 2023.
[49] P. Gokhale, T. Tomesh, M. Suchara, and F. T. Chong, "Faster and more reliable quantum swaps via native gates," *arXiv preprint arXiv:2109.13199*, 2021.
[50] J. Koch, M. Y. Terri, J. Gambetta, A. A. Houck, D. I. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf, "Charge-insensitive qubit design derived from the cooper pair box," *Physical Review A*, vol. 76, no. 4, p. 042319, 2007.
[51] A. Javadi, "Improving quantum circuits with heterogenous gatesets," in *American Physical Society (March Meeting)*, 2023.
[52] G. E. Crooks, "Gates, states, and circuits," *Gates states and circuits*, 2020.
[53] G. Li, Y. Ding, and Y. 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*, 2019, pp. 1001–1014.
[54] A. Li, S. Stein, S. Krishnamoorthy, and J. Ang, "Qasmbench: A low-level qasm benchmark suite for nisq evaluation and simulation," *arXiv preprint arXiv:2005.13018*, 2020.
[55] N. Quetschlich, L. Burgholzer, and R. Wille, "MQT Bench: Benchmarking Software and Design Automation Tools for Quantum Computing," *Quantum*, 2023, MQT Bench is available at https://www.cda.cit.tum.de/mqtbench/.
[56] M. Xia, C. Zhou, C. Liu, P. Patel, X. Cao, P. Lu, B. Mesits, M. Mucci, D. Gorski, D. Pekker *et al.*, "Fast superconducting qubit control with sub-harmonic drives," *arXiv preprint arXiv:2306.10162*, 2023.
[57] J. Liu, P. Li, and H. Zhou, "Not all swaps have the same cost: A case for optimization-aware qubit routing," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 709–725.
[58] J. Liu, E. Younis, M. Weiden, P. Hovland, J. Kubiatowicz, and C. Iancu, "Tackling the qubit mapping problem with permutation-aware synthesis," *arXiv preprint arXiv:2305.02939*, 2023.

A. Artifact Appendix

A.1 Abstract

This artifact accompanies our research on optimizing quantum transpilation by integrating layout and routing stages with gate decomposition. The key strategy involves the use of mirror gates, enabling more cost-efficient routing without increasing decomposition costs. The artifact includes the source code for the Mirage algorithm, Jupyter notebooks detailing our experiments, and datasets used to validate our methodology.

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

- **Program: Python, Jupyter Notebooks**
- **Run-time environment: Python 3.9, Jupyter kernel**
- **OS: Ubuntu 22.04.2 LTS on Windows 10 x86_64**
- **Execution: Quantum circuit transpilation**
- **Output: Circuit depth and swap count analysis**
- **Disk space required (approx.): 1 GB**
- **Time to prepare workflow (approx.): 15 minutes**
- **Time to complete experiments (approx.): 3-4 hours**
- **Publicly available?: Yes**
- **Workflow framework used?: Jupyter notebook**
- **Archived?: 10.5281/zenodo.10208067**

A.3 Description

A.3.1 How to access

The artifact, including the Mirage algorithm and experimental notebooks, is available on GitHub: `https://github.com/Pitt-JonesLab/mirror-gates`. Additionally, the artifact can be found permanently hosted on Zenodo: 10.5281/zenodo.10208067.

A.3.2 Hardware dependencies

The experiments can be conducted on a standard computer setup; however, the artifact has only been tested on Ubuntu.

A.4 Software dependencies

Software dependencies are specified in the 'pyproject.toml' file. Key dependencies include Python 3.9+ and libraries such as 'scipy', 'qutip', 'numpy', 'qiskit', and others. Necessary for the artifact's functionality are the core dependencies:

- 'monodromy' (from `https://github.com/evmckinney9/monodromy.git`)
- 'transpile_benchy' (from `https://github.com/evmckinney9/transpile_benchy.git`)

Note: Monodromy requires 'lrslib', not installed by 'make init'. Install 'lrslib' separately via 'sudo apt install lrslib'. More details at `https://cgm.cs.mcgill.ca/~avis/C/lrs.html`.

The 'init' target in the Makefile handles the environment setup and main dependencies installation. For development, the 'dev-init' target includes additional tools and sets up 'transpile_benchy' and 'monodromy' in editable mode.

The Mirage algorithm is detailed in 'src/mirror_gates/mirage.py', with custom Qiskit transpiler layout and routing entry points defined in 'pyproject.toml'.

A.4.1 Data sets

The experiments utilize custom quantum circuit benchmarks, defined in '.txt' files located in 'src/mirror_gates/circuits/', focusing on comparing circuit depth and swap counts across different topologies. Benchmarks are sourced from QASMBench and MQT-Bench, interfaced via 'transpile_benchy'.

A.5 Installation

To reproduce the experiments:

1. Clone the repository:

```
git clone https://github.com/Pitt-JonesLab/mirror-gates
```

2. Navigate to the cloned directory and run:

```
make init
```

This sets up the environment and installs all required dependencies, (not including 'lrslib' which is a prerequisite for the Monodromy dependency).

A.6 Experiment Workflow

This section outlines the structured process for conducting experiments using the provided Jupyter notebooks. Be sure to select the jupyter notebook kernel source to be the virtual environment (`.venv/`) created by the `make init` command.

A.6.1 Monodromy Mirror Gates Analysis

- **Filename:** `weyl_mirrors/01_weyl_mirrors.ipynb` - Conducts Monte Carlo fitting of coverage volumes using a trial-and-error decomposition ansatz (Figures 3-5).

A.6.2 Key Benchmark Notebooks

- **Filename:** `01_HH_bench.ipynb` - Evaluates $\sqrt{\texttt{iSWAP}}$ on heavy-hex topology (Fig. 12a).
- **Filename:** `02_SL_bench.ipynb` - Analyzes $\sqrt{\texttt{iSWAP}}$ on square-lattice topology (Fig. 12b).
- **Filename:** `r03_HH_CX_bench.ipynb` & `r04_SL_SYC.ipynb` - Investigates CNOT on heavy-hex and SYC on square-lattice, respectively (Figs. 13a and 13b).

A.6.3 Supplementary Experiments

- **Filename:** `03_aggressions.ipynb` - Studies different aggression settings in transpilation (Fig. 10).
- **Filename:** `02_post_selection.ipynb` - Focuses on post-selection analysis for SWAP counts versus Depth (Fig. 11).
- **Filename:** `06_speedups.ipynb` - Compares the performance speed of Mirage against Python SABRE (Fig. 15).

A.6.4 Noisy Simulation Branch Analysis

- **Filename:** `simulation/03_fid_bench.ipynb` - Runs circuits for data collection over fidelity benchmarks.
- **Filename:** `simulation/04_scaling.ipynb` - Addresses scaling benchmarks in the `noisy_simulation` branch (Fig. 14).

A.7 Methodology

Submission, reviewing, and badging methodology:

- `https://www.acm.org/publications/policies/artifact-review-badging`
- `http://cTuning.org/ae/submission-20201122.html`
- `http://cTuning.org/ae/reviewing-20201122.html`