This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2026.3693628

QDP: Worst-case Fidelity-aware Qubit Mapping and Routing using Dynamic Programming

Shui Jiang, Wen Cheng, Yi-Hua Chung, Tsung-Yi Ho *Fellow, IEEE*, and Tsung-Wei Huang

***Abstract*—Qubit mapping and routing (QMR) is a critical step in quantum computing because it maps logical qubits to a physical layout while properly routing all gates. In QMR, achieving high qubit fidelity is essential as it ensures reliable execution of a quantum circuit on a quantum computer. However, existing QMR algorithms focus on optimizing the average fidelity (AVF) across all qubits, ignoring the *worst-case fidelity* (WCF) of individual qubits. In practice, a particularly low-fidelity qubit can greatly diminish the reliability of a quantum circuit as a single qubit failure can lead to the failure of the entire circuit. To address this issue, we introduce QDP, a parallel dynamic programming (DP)-based QMR algorithm that optimizes the WCF while balancing the overall AVF. QDP leverages a state graph as the optimal substructure for our DP formulation to capture all valid quantum circuit execution orders. To enhance performance, QDP introduces a parallel DP-based state traversal algorithm that maximizes WCF through remapping and reduces the size of the state graph via progressive state expansion. Compared with state-of-the-art QMR algorithms, QDP can enhance the WCF by up to 33.88% while achieving comparable AVF on real quantum computers with up to 53 qubits.**

***Index Terms*—Quantum computing, Dynamic programming, Qubit mapping and routing, Qubit fidelity, Parallel algorithm**

I. Introduction

QUBIT mapping and routing (QMR) is critical for executing a logical quantum circuit on a physical quantum computer, as it assigns logical qubits to physical qubits and properly routes all quantum gates. Specifically, a gate on two logical qubits must be placed in adjacent physical qubits to establish reliable entanglement routes [1]. When the two logical qubits are not adjacent, additional `swap` gates must be inserted to iteratively bring them to two adjacent physical qubits. However, these additional `swap` gates can degrade the fidelity of physical qubits and reduce the overall reliability of execution. Therefore, the major goal of QMR is to minimize the number of `swap` gates while satisfying the connectivity constraints of physical qubits. Nevertheless, QMR has been proven to be NP-complete [2], which makes it challenging to find a high-fidelity QMR solution. To address this challenge, existing QMR works have explored various strategies, such as heuristic-based search algorithms [3]–[5] and Boolean optimization techniques [6], [7].

While these strategies can improve qubit fidelity, they primarily focus on optimizing the *average fidelity* (AVF) across all qubits, overlooking a more important metric: the

Manuscript created in November 2025, and revised in March 2026.

Shui Jiang and Tsung-Yi Ho are with the Chinese University of Hong Kong. E-mail: {sjiang22, tyho}@cse.cuhk.edu.hk.
Wen Cheng is with Synopsys Inc. E-mail: chengwen@synopsys.com.
Yi-Hua Chung and Tsung-Wei Huang are with University of Wisconsin-Madison. E-mail: {yihua.chung, tsung-wei.huang}@wisc.edu.

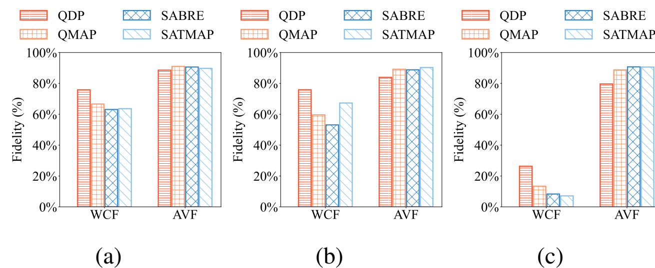


Fig. 1: WCF and AVF results of QDP compared with state-of-the-art QMR algorithms, QMAP, SABRE, and SATMAP on three real IBM quantum computers: (a) 15-qubit Melbourne, (b) 20-qubit Tokyo, and (c) 53-qubit Rochester. QDP significantly improves WCF while achieving comparable AVF to other methods.

worst-case fidelity (WCF) of individual qubits. Specifically, a single qubit with particularly low fidelity can significantly reduce the overall reliability of a quantum circuit, as the failure of just one qubit may cause premature failure of the entire circuit [8]. Unfortunately, this type of WCF-aware QRM problem has been rarely addressed by existing QMR algorithms [1]–[43]. As an example, Figure 1 shows the WCF and AVF values of three state-of-the-art QMR algorithms, QMAP [29], SABRE [1], and SATMAP [6] on three widely used IBM quantum computers: 15-qubit Melbourne, 20-qubit Tokyo, and 53-qubit Rochester. On Melbourne (Figure 1a), although these algorithms achieve AVF values of 89.74%–91.01%, their WCF values are only 63.09%–66.64%. Similar results appear on Tokyo and Rochester too (Figure 1b).

To address this issue, we focus on the *WCF-aware QMR* (WQMR) problem. Unlike existing AVF-centric QMR (AQMR) problems, WQMR exhibits unique challenges. Specifically, optimizing AVF alone, such as the resulting `swap` count or circuit depth, does not necessarily improve WCF. Two mappings may have the same `swap` count and similar AVF, yet exhibit very different WCF if one repeatedly routes gates through a frequently used qubit. This leads to uneven gate distribution and reduces WCF. Additionally, the search space of WQMR is much larger than that of AQMR, as circuits that achieve similar AQMR objectives can still exhibit widely varying WCF values. As a result, these challenges make it difficult to apply existing AQMR solvers to WQMR.

Despite these challenges, we observe a similarity between WQMR and the classic Traveling Salesman Problem (TSP) [44], which can be effectively solved using dynamic programming (DP). Intuitively, both problems aim to find an optimal sequence (qubit mappings in WQMR vs. travel

Authorized licensed use limited to: University of Wisconsin. Downloaded on May 16,2026 at 20:57:39 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2026.3693628

routes in TSP) to optimize a target objective (WCF vs. route distance). This observation inspires us to design a DP-based algorithm called *QDP* to solve WQMR. We summarize our technical innovations below:

- We introduce a *state graph* as the optimal substructure for our DP formulation, capturing all valid quantum circuit execution orders.
- We introduce a parallel DP-based *state traversal algorithm* that dynamically constructs the state graph using multiple threads. It maximizes WCF through remapping and reduces the size of the state graph via progressive state expansion.
- We introduce a shortest path-based *remapping algorithm* to generate candidate qubit mappings and `swap` sequences that can maximize WCF while balancing AVF.
- We introduce a *progressive state expansion algorithm* that incorporates all executable gates into a state only at the time of remapping, thereby significantly reducing the size of the state graph to scale WQMR to larger circuits.

We evaluate the performance of QDP on a set of commonly used QMR benchmarks atop three representative IBM quantum computers: Melbourne, Tokyo, and Rochester. Compared with three state-of-the-art QMR algorithms, SABRE [1], QMAP [29] and SATMAP [6], QDP improves WCF by up to 12.75%, 22.77%, and 19.16%, with only a 2.02%, 4.88%, and 10.97% reduction in AVF on Melbourne, Tokyo, and Rochester, respectively. These results are shown in Figure 1. We plan to integrate QDP into IBM's Qiskit toolchain to advance WCF-aware QMR research.

II. Background

In this section, we formally define the QMR problem (Section II-A) and discuss prior QMR algorithms (Section II-B).

A. Qubit Mapping and Routing

A quantum circuit G is a sequence of quantum gates g_i (i.e., $G = (g_0, g_1, \cdots, g_{|G|-1})$). A quantum gate can perform on one qubit (i.e., single-qubit gate) or two qubits (i.e., two-qubit gate). For example, in Figure 2a, the quantum circuit has three quantum gates on four logical qubits (i.e., $q[0]$, $\cdots$, $q[3]$). In that circuit, `cx` is a two-qubit gate. The execution of a two-qubit gate must satisfy the hardware constraint of a physical quantum computer. A quantum computer architecture consists of physical qubits, which are connected in edges to form a *coupling graph*. For example, there are four physical qubits (i.e., $p_0, \cdots, p_3$) in Figure 2b, forming a coupling graph with three edges $((p_0, p_1), (p_1, p_2), (p_2, p_3))$.

To execute a two-qubit gate, the logical qubits involved must be mapped to adjacent physical qubits. For example, when executing the first gate (i.e., `cx q[0], q[2]`) in Figure 2a on the coupling graph shown in Figure 2b, a feasible solution to map the logical qubits to physical qubits is $\pi_0 : (p_0, p_1, p_2, p_3) \mapsto (q[0], q[2], q[1], q[3])$. In this way, logical qubits $q[0]$ and $q[2]$ are assigned to physical qubits p_0 and p_1, which are adjacent. The process of assigning logical qubits to physical qubits is called *qubit mapping*. π_0 is called an *initial mapping* since it is the first mapping, before the execution of the first quantum gate in Figure 2a.

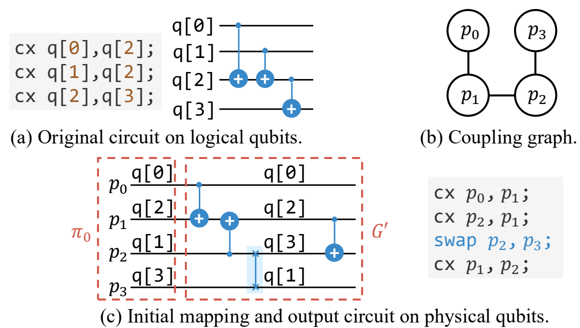


Fig. 2: A QMR example that maps the quantum circuit in (a) to the coupling graph in (b), and results in a quantum circuit on physical qubits in (c).

An initial mapping alone is not sufficient for executing a complete quantum circuit on a quantum computer. As gate execution progresses, the initial mapping might not satisfy later gates. For example, gate `cx q[2], q[3]` in Figure 2a cannot run on mapping $\pi_0 : (p_0, p_1, p_2, p_3) \mapsto (q[0], q[2], q[1], q[3])$, because $q[2]$ and $q[3]$ are mapped to non-adjacent qubits p_1 and p_3. To resolve this, we insert a `swap` gate between p_2 and p_3 to swap their corresponding logical qubits (highlighted in light blue in Figure 2c). As a result, the insertion of this `swap` leads to a new mapping, $(p_0, p_1, p_2, p_3) \mapsto (q[0], q[2], q[3], q[1])$. This adjustment allows the gate `cx q[2], q[3]` to be executed. The process of inserting `swap` gates to adjust qubit mappings, enabling the execution of the original quantum circuit on a quantum computer, is called *qubit routing*.

To summarize, given an input mapping (e.g., π_0 in Figure 2c), QMR determines a gate sequence (e.g., G' in Figure 2c), including necessary `swap`s, that allows a quantum circuit to be executed on a quantum computer. There can be multiple solutions to a QMR problem. Different QMR algorithms determine optimal solutions based on different objectives, such as the number of `swap` gates [1], [6], [9], [29] or circuit depth [1], [5], [10].

B. Related Works

There has been extensive research [1]–[43] on QMR, also known as quantum circuit transpilation or compilation, qubit allocation, and quantum layout synthesis. Due to the NP-completeness of QMR, researchers have proposed various heuristics to solve it [1], [3]–[5], [9], [12], [14], [16], [24], [26], [28], [30], [40], [43]. For instance, [9], [16] introduce look-ahead strategies to evaluate possible qubit movements and minimize the number of `swap` insertions. Approaches such as [3], [4] employ $A^\star$ search to optimize both quantum circuit reliability and execution time. Li et al. [1] use the bidirectional search with nearest neighbor-based heuristic cost (NNC) to minimize the number of `swap` gates. Sinha et al. [5] use Monte Carlo tree search, aided by a graph neural network that evaluates action probabilities, to minimize the resulting quantum circuit depth. Liu et al. [12] optimize QMR for multiple quantum programs in a cloud environment using task

Authorized licensed use limited to: University of Wisconsin. Downloaded on May 16,2026 at 20:57:39 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2026.3693628

TABLE I: Frequently used notations in this paper.

Notation	Description of the notation
G	Quantum circuit on logical qubits.
g	Gate on logical qubits, $g \in G$. g's name and qubits are g.name and $g.q$, respectively.
G'	Quantum circuit on physical qubits, after QMR.
g'	Gate on physical qubits, $g' \in G'$. g''s name and qubits are g'.name and $g'.p$, respectively.
p_i	Physical qubit i.
$q[i]$	Logical qubit i.
n	Number of logical qubits in a quantum circuit.
N	Number of physical qubits in a quantum computer.
$\|G\|$	Number of gates in a quantum circuit G.
F	Fidelity vector, recording fidelity on all physical qubits.
D	DP table that keeps track of different states and the corresponding physical-to-logical mappings.
s	A state in a state graph.
π	A physical-to-logical qubit mapping.
$D[s]$	Dictionary that associates s's each mapping π_i with π_i's properties. The gate sequence and fidelity vector of π_i at s are $D[s][\pi_i]$.seq and $D[s][\pi_i].F$, respectively.
$f(g')$	Calculates the fidelity (i.e., success rate) of gate g'.
$U(F,\theta)$	Calculates the resulting fidelity vector of all physical qubits from a gate sequence θ, given the initial fidelity vector F.
t	Number of threads

scheduling and NNC. These heuristic approaches are often greedy-based, leading to sub-optimal solutions.

To avoid being trapped in sub-optimal solutions, researchers also explore constraint-based search algorithms to find the exact QMR solution under certain conditions [6]–[8], [10], [11], [13], [17], [27], [37]. For instance, Molavi et al. [6] formulate QMR as a maximum satisfiability (SAT) problem with relaxed constraints to minimize the number of swaps. Tan et al. [10] formulate QMR as a satisfiability modulo theories (SMT) optimization problem, to optimize fidelity, the number of swaps, and circuit depth. Bhattacharjee et al. [17] use integer linear programming (ILP) to optimize multiple QMR objectives simultaneously. Nannicini et al. [11] model QMR as a binary integer programming (BIP) problem using a network flow formulation with binary variables to optimize circuit fidelity via minimizing error rate and cross-talk. Shaik et al. [8] model QMR as a classical planning problem, which can scale to quantum circuits with up to nine logical qubits. Since QMR is NP-complete [2], finding the exact solution often costs significant runtime and memory.

While the aforementioned studies improve qubit fidelity, they primarily focus on the AVF of all qubits, overlooking the WCF of individual qubits. As shown in Figure 1, AVF-optimized solution does not necessarily translate to WCF-optimized one, whereas a WCF-aware solution can greatly enhance WCF with comparable AVF. While AVF is a useful metric, WCF is often more critical, as the failure of a single qubit can cause the entire circuit to fail prematurely.

III. Algorithm

In this section, we discuss the details of QDP. We formulate WQMR in Section III-A, and Table I gives frequently used notations throughout this paper along with their descriptions.

Figure 3 gives an overview of our algorithm. Initially, we analyze the quantum circuit and construct a gate dependency graph [1], [45]. From this graph, we then dynamically construct a *state graph* and traverse it to maximize WCF (Section III-B). The state graph is the optimal substructure of our DP formulation, capturing all valid quantum circuit execution orders. In the state graph, each node (i.e., state) is a set of executed quantum gates, following the topological order of the dependency graph. Edges between states denote state transitions (e.g., state A to B in Figure 3), where we perform the following steps: (1) *Remapping*: To accommodate a newly added gate that cannot be executed on the current mapping, we remap the circuit. Our remapping algorithm (Section III-D) identifies the optimal swap paths on the target architecture while balancing WCF and AVF, and then generates candidate mappings with their corresponding swap sequences. (2) *Expansion*: Upon remapping, we expand the state further to include all gates executable under the new mapping (Section III-C). This can significantly reduce the size of the state graph to scale WQMR to larger circuits. State graph traversal can be parallelized for independent states (e.g., states A and C), with each thread handling independent state transitions.

Starting from the initial state, we traverse the state graph until reaching the final state where all gates are executed (e.g., state D).

A. WCF-aware QMR Problem Formulation

To understand the importance of WCF, we show that it defines a fundamental upper bound on circuit output correctness. Consider any measured bitstring in a quantum circuit and define event E as "the bitstring is correct". Let E_i denote the event that qubit i is correct for this bitstring. Since a bitstring is correct only if every qubit is simultaneously correct (i.e., $E = \bigcap_i E_i$), we have:

$$\Pr(E) = \Pr(\bigcap_i E_i) \leq \min_i \Pr(E_i) = \text{WCF}. \quad (1)$$

This shows that WCF imposes a hard ceiling on the probability that any particular output bitstring is correct. In contrast, a higher AVF does not prevent a single weak qubit from severely limiting $\Pr(E)$. Hence, improving WCF directly raises this ceiling and mitigates weakest-qubit bottlenecks in measured outcomes.

In this paper, fidelity for each physical qubit p_i is modeled as *success rate*, a commonly used metric in existing QMR solvers [17], [25], [26]. It is calculated by multiplying the success rates of all gates g'_j applied to p_i (i.e, $p_i \in g'_j.p$). Therefore, after executing a gate sequence $G' = (g'_0, g'_1, \cdots, g'_{|G'|-1})$, the fidelity for physical qubit p_i is $\prod_{g'_j \in G', p_i \in g'_j.p} f(g'_j)$. We consider the fidelity of a two-qubit gate g'_{two} (excluding swap) to be $f(g'_{two}) = 99\%$ and that of a single-qubit gate g'_{single} to be $f(g'_{single}) = 99.9\%$, which is typical of real quantum hardware [13], [26]. Note that these fidelities are parameterized, not fixed constants. QDP can be easily reconfigured to adapt to higher fidelities (e.g., ion/neutral-atom architectures at 99.9%–99.99%) or lower fidelities (e.g., $\sim$ 95%). Additionally, the fidelity of a swap gate is $(f(g'_{two}))^3$, because a swap gate consists of three

Authorized licensed use limited to: University of Wisconsin. Downloaded on May 16,2026 at 20:57:39 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2026.3693628

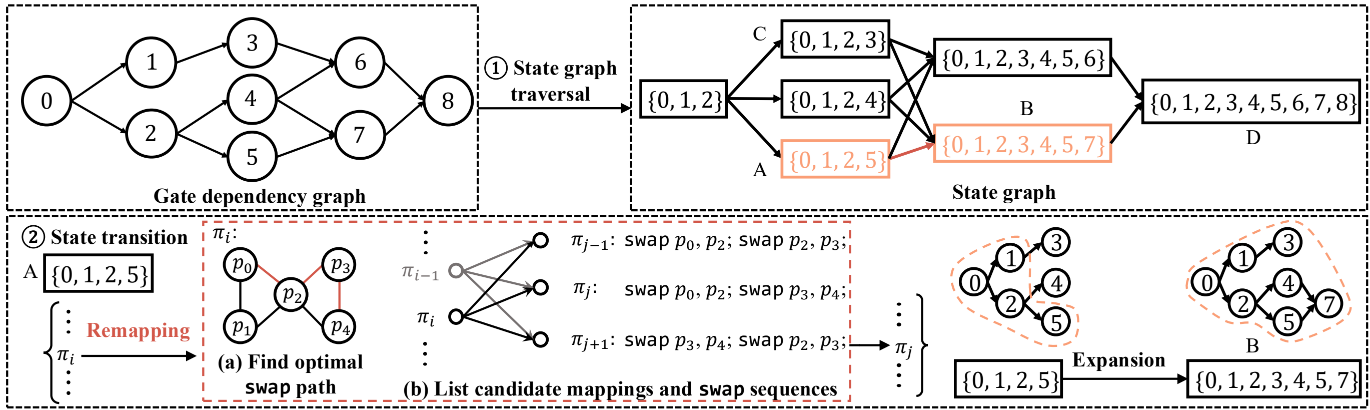


Fig. 3: The overview of QDP algorithm. Step ① dynamically constructs and traverses a state graph based on the gate dependency graph's topological order. In step ②, we transition between adjacent states using our state transition algorithm. This includes *remapping* the circuit to accommodate a new gate and *expanding* a state to include all gates executable under the new mapping.

two-qubit `cx` gates. WCF (AVF) is defined as the minimum (average) across physical qubits, where each qubit computes the product of fidelities of all gates involving that qubit.

We map an original quantum circuit G to a circuit G' that can run on a specific quantum computer architecture by changing the gate sequence and adding `swap`s. Given an initial mapping π_0, our primary objective is to find an optimal G' (i.e., $G'^{\star}$) that maximizes WCF after G''s execution on the target architecture. This optimization problem can be expressed in Equation 2.

$$G'^{\star} = \underset{G'=(g'_0, g'_1, \cdots, g'_{|G'|-1})}{\arg\max} \; \min_{p_i} \{ \prod_{g'_j \in G',\; g'_j \text{ targets } p_i} f(g'_j) \} \quad (2)$$

The optimization is subject to two constraints: (1) **Physical constraint:** G' must be executable on the target architecture's coupling graph. This is guaranteed by selecting only the edges on the coupling graph for `swap` gates. (2) **Logical constraint:** G' must remain functionally equivalent to G. We can guarantee this by traversing only along the edges of the state graph, as it captures all valid execution orders of G.

B. State Graph Traversal

DP table

States	Qubit mapping	(Gate sequence, Fidelity)
s_0	π_0	(seq_0, F_0)
	π_1	(seq_1, F_1)
	π_2	(seq_2, F_2)
s_1	π_3	(seq_3, F_3)
	π_4	(seq_4, F_4)
s_2	π_5	(seq_5, F_5)

Fig. 4: The organization of a DP table.

To create an optimal substructure of our DP, we construct a state graph that captures all valid execution orders of a quantum circuit. Algorithm 1 presents a parallel DP-based *state traversal algorithm* that dynamically constructs the state graph while maximizing WCF. The inputs consist of: (1) H, the gate dependency graph and (2) π_0, the initial qubit mapping, efficiently generated using QMAP [29]. At line 1, starting from the initial empty state (line 1), we progressively expand it under mapping π_0 to generate state s_0 and frontier set f_0 (Section III-C). The frontier set f_0 contains gates that cannot yet be executed under mapping π_0 and have direct dependencies on gates in s_0, similar to to D-frontier [46] in the D-Algorithm [47] for Automatic Test Pattern Generation. For example, in Figure 5, expansion from empty state in (a) yields state $s_0 = \{0, 1, 2\}$ in (b), with $f_0 = \{3, 4, 5\}$ blocked under π_0. At lines 2–3, we compute the fidelity F of the physical qubits after executing all gates in s_0. We then store the gate sequence of s_0 and its fidelity F in a DP table D for state s_0 under mapping π_0. As illustrated in Figure 4, a DP table is an unordered map that associates each state with an inner qubit mapping table. This inner qubit mapping table is another unordered map, which links each qubit mapping to a tuple containing the gate sequence required to reach that state and its corresponding fidelity vector F. Next, we maintain a bucket list B that maps the number of gates in a state to the set of states. At line 4, we add s_0 to $B[\text{len}(s_0)]$. This data structure enables *parallel processing*: states with the same number of gates (i.e., mapped to the same bucket in B) are independent and thus can be processed in parallel, as they share no dependencies. For example, the states $\{0, 1, 2, 3\}$, $\{0, 1, 2, 4\}$, and $\{0, 1, 2, 5\}$ (Figure 5c) are in the same bucket and are thus mutually independent, enabling parallel processing.

Next, we process all states $s_i \in b$ using multiple threads for each bucket $b \in B$ (lines 5–7). For each s_i, we obtain its frontier set f_i (line 8) and retain only the top-k entries in $D[s_i]$ ranked by WCF (line 9). This pruning is essential for scalability: as circuit depth grows, QDP's runtime and memory usage increase due to the exponential growth of mappings per state. However, we observe that many of these mappings,

Authorized licensed use limited to: University of Wisconsin. Downloaded on May 16,2026 at 20:57:39 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2026.3693628

particularly those with low WCF, have negligible impact on the final result. For example, when solving QMR for circuit 4gt13_90 [29] on IBM Melbourne, QDP can prune 90% of mappings without significantly affecting the final WCF.

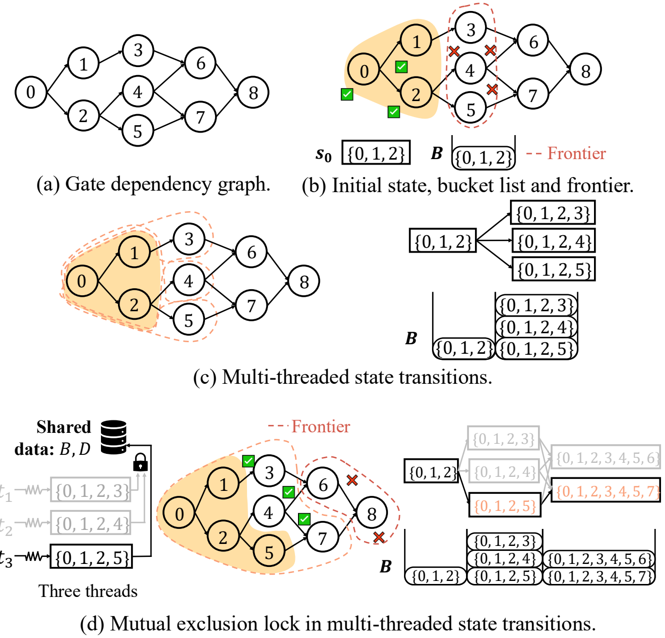
Algorithm 1 State graph traversal

Require: H, gate dependency graph; π_0, initial qubit mapping.

1: $s_0, f_0 \leftarrow$Expand$(\pi_0, \{\})$ // Section III-C
2: $F \leftarrow$ Fidelity after executing gates in s_0
3: $D[s_0][\pi_0] \leftarrow$ {seq: Sequence$(s_0), F : F\}$
4: $B[\text{len}(s_0)] \leftarrow \{s_0\}$
5: **for** Bucket b in B **do**
6:   **#pragma omp parallel for**
7:   **for** $s_i \in b$ **do**
8:     $f_i \leftarrow$ frontier of s_i
9:     Top_k_WCF$(D[s_i])$
10:     **for** $g \in f_i$ **do**
11:       $s_j \leftarrow s_i \cup \{g\}$
12:       **for** $\forall \pi_i$ at s_i **do**
13:         $\Pi_j, \Theta_{ij} \leftarrow$ Remap(π_i, g) // Section III-D
14:         **for** $\pi_j, \theta_{ij} \in \Pi_j, \Theta_{ij}$ **do**
15:           $s_e, f_e \leftarrow$Expand(π_j, s_j)
16:           $\theta_{ie} \leftarrow$ Combine$(\theta_{ij}$, Sequence$(s_e \setminus s_j))$
17:           /* DP table update. */
18:           **#pragma omp critical**
19:           {
20:           $B[\text{len}(s_e)].\text{push}(s_e)$
21:           **if** π_e is not recorded at s_e **or** $\min\{U(D[s_i][\pi_i].F, \theta_{ie})\} > \min\{D[s_e][\pi_j].F\}$ **then**
22:             $D[s_e][\pi_j] \leftarrow$ {seq: Combine$(D[s_i][\pi_i]$.seq $, \theta_{ie}), F : U(D[s_i][\pi_i].F, \theta_{ie})\}$
23:           }

Then, we iterate through every gate g in the frontier set f_i (line 10). For each g, we first construct a temporary state s_j by adding g to the current state s_i (line 11). Since g cannot be directly executed under the current mapping π_i due to hardware constraints, we remap π_i by inserting additional `swaps` on the target architecture (lines 12–13). Using our remapping algorithm (Section III-D), we generate candidate mappings Π_j and the corresponding gate sequences Θ_{ij}, which include the required `swaps`. For each candidate mapping $\pi_j \in \Pi_j$ and its associated gate sequence $\theta_{ij} \in \Theta_{ij}$ (line 14), we further expand s_j to include all gates that become executable under π_j (line 15), and update the gate sequence to reach s_e from s_i (line 16).

Then, in lines 17–23, we update the bucket list B with s_e and the DP table D with the gate sequence that maximizes the WCF. Since B and D are shared data structures across all threads, each thread must access them under mutual exclusion to prevent race conditions [48]. We enforce this by wrapping accesses in an OpenMP critical section using `#pragma omp critical` [49]. For example, in Figure 5d, when thread t_3 transitions from state $\{0, 1, 2, 5\}$ to state $\{0, 1, 2, 3, 4, 5, 7\}$, threads t_1 and t_2 are blocked by the mutual exclusion lock.



(a) Gate dependency graph. (b) Initial state, bucket list and frontier.

(c) Multi-threaded state transitions.

(d) Mutual exclusion lock in multi-threaded state transitions.

Fig. 5: Parallel DP-based state graph traversal algorithm.

Note that the order in which threads access the shared DP table D can vary. If we sort mappings only by WCF and retain the top-k mappings in D (line 9), mappings with identical WCF values may or may not be preserved depending on thread order. As a result, the algorithm's output can vary across runs. To eliminate this run-to-run variability, we impose a deterministic total ordering over candidate mappings when WCF ties occur, and we always prune according to this ordering. We incorporate these tie-breakers into operator Top_k_WCF (line 9). Concretely, we rank mappings using the following tie-breakers in strict priority order:

1) WCF (descending),
2) Number of physical qubits used so far (ascending),
3) Maximum number of gates applied to any physical qubit so far (ascending),
4) AVF (descending),
5) Lexicographical order of the mapping (final deterministic tie-breaker).

This order prioritizes WCF while penalizing the physical qubit count, over-utilization of physical qubit, and AVF degradation. The tie-breakers ensure that, even when parallel traversal discovers candidates in different orders, the retained top-k set is identical across runs.

Next, to update the DP table D, for each expanded state s_e and its mapping π_j, we consider all possible gate sequences θ_{ie} that lead to s_e. Among these, we select the best sequence $\theta^\star$ that maximizes the WCF, as defined in Equation 3.

$$\theta^\star = \arg\max_{\theta_{ie}} \min_{p_0, \cdots, p_{N-1}} \{U(D[s_i][\pi_i].F, \theta_{ie})\} \quad (3)$$

In Equation 3, $U(F, \theta)$ calculates the updated fidelity vector after applying gate sequence θ to fidelity vector $F = (F_0, \cdots, F_{N-1})$. Formally, we define $U(F, \theta)$ in Equation 4.

Authorized licensed use limited to: University of Wisconsin. Downloaded on May 16,2026 at 20:57:39 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2026.3693628

$$U(F,\theta) = (F_0 \prod_{\substack{g'_i \in \theta \\ g'_i \text{ targets } p_0}} f(g'_i), \cdots, F_{N-1} \prod_{\substack{g'_i \in \theta \\ g'_i \text{ targets } p_{N-1}}} f(g'_i)) \tag{4}$$

For example, if we apply a two-gate sequence ((cx, p_1, p_2), (swap, p_2, p_3)) to a fidelity vector $(1, 0.99, 0.98, 1)$, the resulting fidelity vector is $(1, 0.99 \times 0.99, 0.98 \times 0.99^{(1+3)}, 0.99^3) = (1, 0.98, 0.94, 0.97)$, since one swap consists of three cxs.

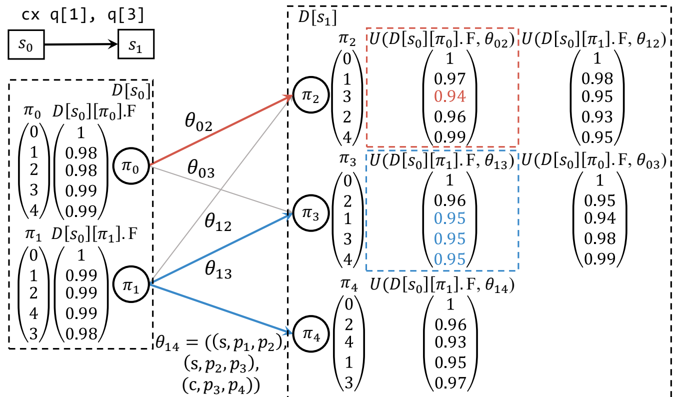


Fig. 6: Illustration of a DP table update. We update $D[s_1]$ from $D[s_0]$, where the gate cx q[1], q[3] triggers the transition from state s_0 to s_1. In gate sequence θ_{14}, "s" and "c" represent swap and cx gates, respectively.

Figure 6 shows an example of a DP table update. We update $D[s_1]$ from $D[s_0]$, where the gate cx q[1], q[3] triggers the transition from state s_0 to s_1. In $D[s_0]$, two mappings, π_0 and π_1, are shown along with their associated fidelity vectors. π_0 has two candidate mappings: π_2 and π_3; π_1 has three: π_2, π_3, and π_4. Mapping π_4's fidelity vector can be directly computed using $U(D[s_0][\pi_1].F, \theta_{14}) = (1, 0.99 \times 0.99^3, 0.99 \times 0.99^6, 0.99 \times 0.99^4, 0.98 \times 0.99) = (1, 0.96, 0.93, 0.95, 0.97)$. Since π_1 in state s_0 is the only predecessor of π_4, we append θ_{14} to π_4's gate sequence ($D[s_1][\pi_4]$.seq) and update π_4's fidelity vector ($D[s_1][\pi_4].F$) as $U(D[s_0][\pi_1].F, \theta_{14})$. Mappings π_2 and π_3 each have two parent mappings: π_0 and π_1. In this case, π_2 and π_3 need to determine which parent leads to the highest WCF. For π_2, we compare $U(D[s_0][\pi_0].F, \theta_{02}) = (1, 0.97, 0.94, 0.96, 0.99)$ and $U(D[s_0][\pi_1].F, \theta_{12}) = (1, 0.98, 0.95, 0.93, 0.95)$. Since the former has a higher WCF, we append θ_{02} (thick red edge in Figure 6) to $D[s_1][\pi_2]$.seq and update $D[s_1][\pi_2].F$ as $(1, 0.97, 0.94, 0.96, 0.99)$ (red dashed box in Figure 6). A similar update applies to π_3: since $U(D[s_0][\pi_1].F, \theta_{13})$ (blue dashed box) has a higher WCF than $U(D[s_0][\pi_0].F, \theta_{03})$, we append θ_{13} to $D[s_1][\pi_3]$.seq and update $D[s_1][\pi_3].F$ as $U(D[s_0][\pi_1].F, \theta_{13})$ (upper thick blue edge).

C. Progressive State Expansion

The goal of progressive state expansion is to significantly reduce the size of the state graph to enable WQMR in large-scale quantum circuits. While states can be generated by including gates one at a time, this approach results in an excessively large state graph, making it unscalable for large quantum circuits. Moreover, this approach produces many redundant states where the qubit mapping remains unchanged. To address these issues, we expand as many gates as possible in a single state, generating new states only when remapping occurs. As a result, our strategy minimizes redundant states and improves scalability. For instance, the state graph in Figure 7a, constructed by incrementally adding one gate at a time, is 3× larger than that in Figure 7b, which includes states only when remapping takes place.

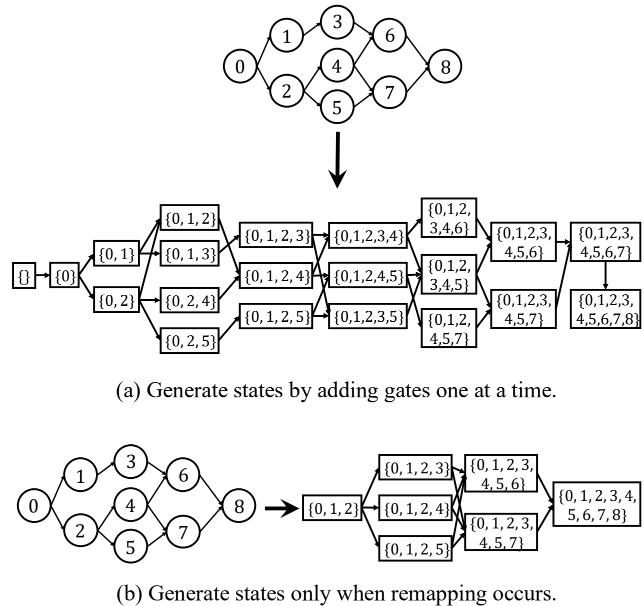


Fig. 7: State graphs under different state-generation policies.

Algorithm 2 presents our progressive state expansion. Given a mapping π and an input state s_i, the algorithm outputs the expanded state s_o, and a frontier set f. Initially, f is set to empty set, the *progress* flag is initialized to true, and s_o to s_i (lines 1-3). While the *progress* flag is true, the algorithm expands by including new gates (line 4). Upon entering the loop, the *progress* flag is set to false, and the algorithm checks whether any gate can trigger further expansion, potentially setting the flag back to true (line 5). Then, we iterate over all gates whose dependencies have been executed but that have not yet been executed themselves (lines 6-8). For instance, in Figure 5d, gate 3 has its dependency (gate 1) executed, since gate 1 is in the current state {0, 1, 2, 5} (shaded yellow), but gate 3 itself has not been executed, making it a candidate for expansion. When a candidate gate g is executable under the current mapping, we add it to s_o and set *progress* flag to true (lines 9-11). Otherwise, we add g to f, as it cannot be executed under the current mapping (lines 12-13).

D. Remapping Algorithm

To accommodate a new gate that cannot be executed under the current qubit mapping, we introduce a remapping algorithm that first identifies the optimal swap path (Section III-D1) between the two qubits involved in the gate. Based

Authorized licensed use limited to: University of Wisconsin. Downloaded on May 16,2026 at 20:57:39 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Algorithm 2 Progressive state expansion

Require: π, mapping; s_i, input state.
Ensure: s_o, expanded state; f, frontier set
1: $f \leftarrow \{\}$
2: $progress \leftarrow$ true
3: $s_o \leftarrow s_i$
4: **while** $progress$ **do**
5: $progress \leftarrow$ false
6: **for** $g \in gates_not_executed$ **do**
7: **if** not all g's dependencies are executed **then**
8: **continue**
9: **if** Executable(g) **then**
10: s_o.push(g)
11: $prgress \leftarrow$ true
12: **else**
13: f.push(g)

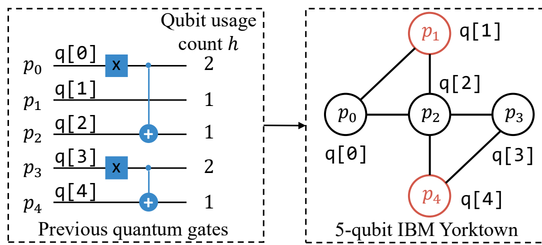


Fig. 8: `swap` path-finding on the 5-qubit IBM Yorktown quantum computer for executing `cx q[1], q[4]`, assuming an identity initial mapping.

on this path, we then generate candidate qubit mappings and `swap` sequences (Section III-D2).

1) Optimal `swap` Path: We define a cost function $C(\cdot)$ in Equation 5 to identify the minimum-cost path on the target architecture, thereby optimizing the `swap` path.

$$C(P) = |P| + \lambda \cdot \frac{\max_{p_i \in P}\{h[p_i]\}}{\max_{\forall p_i}\{h[p_i]\}} \tag{5}$$

Here, P is a path on the coupling graph, h tracks the number of prior quantum gates applied on each physical qubit, and λ is a tunable weighting coefficient. The first term, $|P|$, represents path P's length and corresponds directly to the number of `swaps` required. The second term, $\frac{\max_{p_i \in P}\{h[p_i]\}}{\max_{\forall p_i}\{h[p_i]\}}$, normalizes the maximum qubit usage along P by the global maximum usage across all qubits. The normalization ensures that the second term remains bounded between 0 and 1, regardless of the total number of gates applied prior. Since more `swaps` lowers AVF and frequently reusing the same qubit lowers WCF, these two terms jointly capture the trade-off between AVF and WCF. By adjusting the tunable parameter λ, we can prioritize WCF over AVF, as higher values of λ penalize qubit reuse more strongly.

Figure 8 illustrates the process of finding an optimal `swap` path on the 5-qubit IBM Yorktown quantum computer to execute `cx q[1], q[4]`. Assuming an identity initial mapping (i.e., $(p_0, p_1, p_2, p_3, p_4) \mapsto (q[0], q[1], q[2], q[3], q[4])$), logical qubits $q[1]$ and $q[4]$ are initially mapped to physical qubits p_1 and p_4, respectively. Since p_1 and p_4 are not adjacent in Figure 8, we must find an optimal `swap` path between them. There are four paths between p_1 and p_4: (p_1, p_2, p_4), (p_1, p_0, p_2, p_4), (p_1, p_2, p_3, p_4), and (p_1, p_0, p_2, p_3 p_4). Assuming $\lambda = 5$ in Equation 5, the corresponding costs are 8, 14, 14, and 15, respectively. Thus, the optimal `swap` path is $P = (p_1, p_2, p_4)$.

Inspired by the shortest path algorithm [50], we use Algorithm 3 to find the set of optimal paths $Paths$ between physical qubits src and dst using the cost function in Equation 5. At line 1, We find the global maximum qubit usage h_max across all physical qubits. We initialize a priority queue pq with the source qubit src at its initial cost of 0, and set up a map min_cost to track the minimum cost to reach each qubit, initially empty (lines 2-3). pq is maintained such that entries with lower cost are popped first, following min-heap semantics based on the $cost$ field.

Next, when pq is not empty, we find the top entry and pop it (lines 4-6). If this entry correspond to the destination qubit dst, we add its path to the result set $Paths$ (lines 7-9). Otherwise, for each neighboring qubits nbr of the current entry's qubit, we compute the cost of nbr based on Equation 5 (lines 10-13). If nbr is not yet in min_cost, or if the newly computed cost to reach nbr is lower than the previously recorded cost, we update $min_cost[nbr]$ with this smaller cost, update the path to nbr, and push a new entry for nbr to pq (lines 14-18).

Algorithm 3 Optimal `swap` path finding

Require: src, source physical qubit; dst, destination physical qubit.
Ensure: $Paths$, set of optimal `swap` paths.
1: $h_max \leftarrow$max(h) // h : qubit usage count for prior gates
2: $pq \leftarrow$ PriorityQueue({$cost$: 0, $qubit$: src, $path$: $\{src\}$}) // pq is ordered by $cost$
3: $min_cost \leftarrow \{\}$
4: **while** pq is not empty **do**
5: $node \leftarrow pq$.top()
6: pq.pop()
7: **if** $node == dst$ **then**
8: $Paths$.push($node.path$)
9: **continue**
10: **for** $nbr \in node.qubits$ **do**
11: $usage \leftarrow \max_{p_i \in P \cup \{nbr\}}\{h[p_i]\}$
12: $p_len \leftarrow |node.path|$
13: $cost \leftarrow p_len + \lambda \cdot usage/h_max$
14: **if** $nbr \notin min_cost$.keys() **or** $min_cost[nbr] > cost$ **then**
15: $min_cost[nbr] \leftarrow cost$
16: $p_n \leftarrow node.path$
17: p_n.push(nbr)
18: pq.push($\{cost : cost, qubit : nbr, path : p_n\}$)
19: **return** $Paths$

2) Candidate Generation: After identifying the optimal `swap` path, we insert `swap` gates to remap the circuit from qubit mapping π_0 to accommodate a new gate g. Since there are multiple ways to insert these `swap` gates, this results in several candidate mappings and corresponding gate sequences.

Authorized licensed use limited to: University of Wisconsin. Downloaded on May 16,2026 at 20:57:39 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2026.3693628

Algorithm 4 Candidate mapping and gate sequence generation

Require: π_0, input physical-to-logical qubit mapping; g, new quantum gate to execute on logical qubits;
Ensure: Π, a set of candidate mappings; Θ, a set of candidate gate sequences.

1: $p_\alpha \leftarrow \pi_0$.index$(g.q[0])$; $p_\beta \leftarrow \pi_0$.index$(g.q[1])$
2: $Paths \leftarrow$OptimalSwapPath(p_α, p_β) // Section III-D1
3: $\Pi, \Theta \leftarrow \{\}$
4: **for** $P \in Paths$ **do**
5: $\quad pair \leftarrow (p_\alpha, p_\beta)$; $P_{\backslash pair} \leftarrow P - pair$
6: $\quad$ **for** $i \in \{0, 1, \cdots, |P_{\backslash pair}|\}$ **do**
7: $\quad\quad P_{new} \leftarrow P_{\backslash pair}[0:i] + pair + P_{\backslash pair}[i:|P_{\backslash pair}|]$
8: $\quad\quad \theta_c \leftarrow ()$
9: $\quad\quad$ **for** $k \in$ Combine(range(i), range($|P|-2$, i, -1)) **do**
10: $\quad\quad\quad \theta_c$.push((swap, $P[k]$, $P[k+1]$))
11: $\quad\quad \pi_c \leftarrow \pi_0$
12: $\quad\quad$ **for** $k \in \{0, 1, \cdots, |P|-1\}$ **do**
13: $\quad\quad\quad \pi_c[P[k]] \leftarrow \pi_0[P_{new}[k]]$
14: $\quad\quad \theta_c$.push((g.name, π_c.index$(g.q[0])$, π_c.index$(g.q[1])$))
15: $\quad\quad \Pi$.push(π_c); Θ.push(θ_c)
16: **return** Π, Θ

Algorithm 4 generates these candidate mappings in Π and gate sequences in Θ. Each gate sequence in Θ consists of a swap sequence that prepares a candidate mapping, followed by the new gate applied under that mapping. At line 1, we determine the two physical qubits p_α and p_β of gate g. We then identify optimal swap paths that maximize WCF while balancing AVF (line 2). Next, we generate candidate mappings and gate sequences on each optimal swap path P found in Algorithm 3 (lines 3-4).

At line 5, we construct $pair$ from the two physical qubits p_α and p_β, and exclude $pair$ from P to obtain $P_{\backslash pair}$. For example, given $P = (p_1, p_2, p_4)$ (optimal path in Figure 8) and $pair = (p_1, p_4)$, we have $P_{\backslash pair} = (p_2)$. Algorithm 4 then inserts $pair$ iteratively into different positions relative to the elements in $P_{\backslash pair}$, to generate new order of physical qubits, P_{new} (lines 6-7). In this case, we can insert $pair = (p_1, p_4)$ into $P_{\backslash pair} = (p_2)$ at two locations: before p_2 and after p_2 (i.e., $i = 0, 1$ at line 6), forming two P_{new}s (i.e., (p_1, p_4, p_2) and (p_2, p_1, p_4)). These two P_{new}s lead to two candidate mappings, as shown in Figure 9.

$P = (p_1, p_2, p_4)$ $P_{new} = (p_1, p_4, p_2)$
$P_{\backslash pair} = (p_2)$

(a) $(p_0, p_1, p_2, p_3, p_4) \mapsto (q[0], q[1], q[4], q[3], q[2])$.

$P = (p_1, p_2, p_4)$ $P_{new} = (p_2, p_1, p_4)$
$P_{\backslash pair} = (p_2)$

(b) $(p_0, p_1, p_2, p_3, p_4) \mapsto (q[0], q[2], q[1], q[3], q[4])$.

Fig. 9: Two candidate mappings on the optimal path (p_1, p_2, p_4) in Figure 8. The physical qubits in black, blue, and red dashed boxes constitute P, $P_{\backslash pair}$, and P_{new}, respectively.

For each P_{new}, we first determine the swap sequence required to reorder physical qubits from P to P_{new} (lines 8-10). For example, given the P and $P_{\backslash pair}$ in Figure 9b, the resulting swap sequence is $((\text{swap}, p_1, p_2))$. Logical qubits are then reordered based on the new physical qubit order in P_{new}, creating a candidate mapping π_c (lines 11-13). For example, in Figure 9b, the initial mapping is an identity mapping, i.e., $\pi_0 : (p_0, p_1, p_2, p_3, p_4) \mapsto (q[0], q[1], q[2], q[3], q[4])$. Since $|P| = 3$, we then iterate k through $\{0, 1, 2\}$ at line 12: $\pi_c[P[0]] = \pi_c[p_1] = \pi_0[P_{new}[0]] = \pi_0[p_2] = q[2], \cdots, \pi_c[P[2]] = \pi_c[p_4] = \pi_0[P_{new}[2]] = \pi_0[p_4] = q[4]$. This results in a candidate mapping $\pi_c : (p_0, p_1, p_2, p_3, p_4) \mapsto (q[0], q[2], q[1], q[3], q[4])$. Next, we map gate g to physical qubits on candidate mapping π_c and add it to candidate gate sequence θ_c. We then push π_c and θ_c to Π and Θ (lines 14-15). Finally, we return Π and Θ at line 16.

IV. Experimental Results

In this section, we evaluate the performance of QDP on commonly used QMR benchmarks from QMAP [29]. In addition to the existing QMAP benchmarks, which have ≤ 16 active logical qubits, we construct a larger circuit with more qubits, denoted as "Combo", by combining ex1_226 and 4gt13_90 together with randomly inserted cx gate. This allows us to study QDP's scalability to a circuit with larger numbers of logical qubits, while still keeping the circuit within the QMAP benchmark family. We select three commonly used IBM quantum computer architectures [51], the 15-qubit Melbourne, the 20-qubit Tokyo and the 53-qubit Rochester, as illustrated in Figure 10. Later in this section, we further explore additional architectures derived from Melbourne with varying connectivity. We compare the WCF, AVF, and runtime of QDP against three state-of-the-art, open-source, and highly optimized QMR algorithms, QMAP [29], SABRE [1], and SATMAP [6]. Note that SATMAP fails on circuits rd53_311 and mod5adder_127 due to solver errors (as shown in Table II and Table III). All the experiments are conducted on a Ubuntu 22.04.2 LTS machine with two Intel Xeon Gold 6226R CPUs (64 logical cores in total) at 2.9 GHz and 256 GB memory capacity. All runtimes reported in this paper are measured as wall-clock time. We terminate the runs that take longer than two hours. Unless otherwise specified, we set $\lambda = 20$, $k = 50$ and $t = 8$, as these values are determined to be effective across multiple circuits.

A. Fidelity Comparison

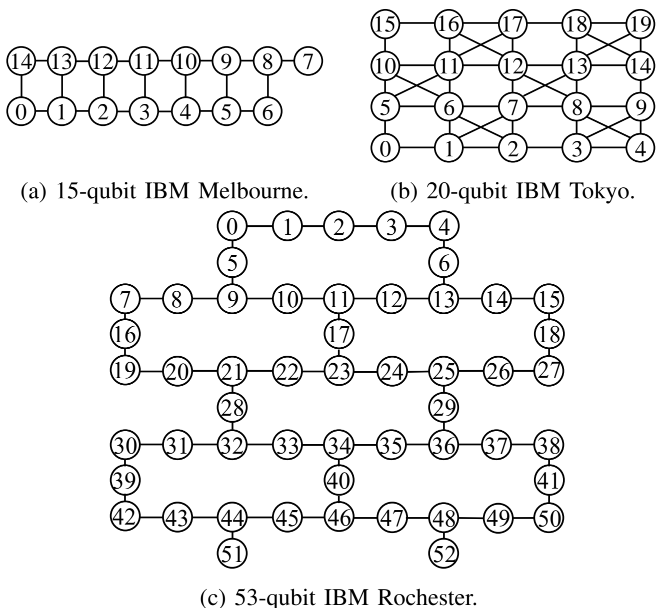
Table II compares the WCF and AVF among QDP, QMAP, SABRE, and SATMAP. Compared with baselines, QDP significantly improves WCF while achieving comparable AVF. For example, on Melbourne, QDP improves WCF by an average of 9.20% compared to QMAP (75.84% vs. 66.64%), with only a 2.42% loss in AVF (88.59% vs. 91.01%). Similarly, on Tokyo, QDP improves WCF by an average of 16.28% compared to QMAP (75.88% vs. 59.60%), with only a 5.25% loss in AVF (83.91% vs. 89.16%). Similar data can be observed in comparison to SABRE and SATMAP. We attribute these results to QDP's strategy of selecting, at each state, the

Authorized licensed use limited to: University of Wisconsin. Downloaded on May 16,2026 at 20:57:39 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edit
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2026.3693628

TABLE II: Comparison of WCF and AVF among QDP (ours), QMAP [29], SABRE [1], and SATMAP [6] on 20 QMR benchmarks. "δ" represents the average number of two-qubit gates per logical qubit.

Architectures		Circuits				WCF				AVF			
Name	N	Name	n	$\|G\|$	δ	QDP	QMAP	SABRE	SATMAP	QDP	QMAP	SABRE	SATMAP
Melbourne	15	rd32-v0_66	4	34	4.00	86.53%	81.38%	75.85%	75.25%	95.59%	96.13%	95.91%	95.03%
		4gt13_90	5	107	10.60	61.92%	56.91%	40.64%	46.83%	85.58%	87.55%	85.92%	86.79%
		ex1_226	6	7	0.83	95.10%	94.15%	94.15%	93.21%	98.54%	98.94%	98.94%	98.81%
		alu-bdd_288	7	84	5.43	74.42%	48.94%	60.93%	51.57%	84.46%	88.24%	88.89%	87.11%
		rd53_138	8	132	7.50	61.24%	51.83%	43.87%	51.36%	78.78%	84.19%	83.40%	80.98%
		Average				75.84%	66.64%	63.09%	63.64%	88.59%	91.01%	90.61%	89.74%
Tokyo	20	mini_alu_305	10	173	7.70	71.92%	62.36%	44.09%	67.64%	82.19%	87.18%	87.59%	87.75%
		9symml_195	11	121	4.82	77.48%	50.59%	55.94%	74.65%	88.48%	90.42%	89.32%	92.24%
		dist_223	13	132	4.69	76.31%	59.78%	58.71%	68.46%	82.53%	88.59%	88.74%	89.18%
		0410184_169	14	82	4.64	77.78%	68.88%	64.78%	64.13%	83.36%	88.69%	88.48%	90.67%
		Combo	17	123	3.82	75.93%	56.40%	42.05%	61.55%	82.99%	90.90%	89.84%	91.54%
		Average				75.88%	59.60%	53.11%	67.29%	83.91%	89.16%	88.79%	90.28%
Rochester	53	ham7_104	7	320	21.28	29.26%	7.92%	5.77%	> 2h	84.30%	91.68%	92.42%	> 2h
		mini-alu_167	5	288	25.20	30.73%	15.47%	7.06%	> 2h	85.12%	92.02%	93.32%	> 2h
		mod5adder_127	6	555	39.83	13.00%	3.05%	2.35%	–	80.57%	90.47%	90.38%	–
		mod10_171	5	244	21.60	37.76%	20.62%	10.74%	14.38%	86.66%	93.79%	94.03%	92.98%
		sf_276	6	778	56.00	7.07%	3.46%	0.31%	0.17%	79.13%	82.89%	90.20%	87.69%
		rd53_131	7	469	28.57	19.75%	8.29%	6.78%	> 2h	82.21%	90.17%	89.65%	> 2h
		rd53_311	13	275	9.54	43.34%	20.27%	21.17%	–	73.69%	90.12%	89.38%	–
		sym6_316	14	270	8.78	44.71%	35.52%	14.12%	> 2h	73.68%	88.66%	90.31%	> 2h
		mod8-10_178	6	342	25.33	27.82%	17.25%	14.73%	6.86%	84.16%	87.72%	91.35%	91.14%
		wim_266	11	986	38.82	9.51%	1.84%	0.28%	> 2h	66.78%	79.99%	86.11%	> 2h
		Average				26.30%	13.37%	8.33%	7.14%	79.63%	88.75%	90.72%	90.60%



(a) 15-qubit IBM Melbourne. (b) 20-qubit IBM Tokyo.

(c) 53-qubit IBM Rochester.

Fig. 10: Three commonly used IBM quantum computer architectures. Node labels indicate physical qubits.

mapping that maximizes WCF during state transitions, while balancing WCF and AVF during `swap` path selection. These results highlight the effectiveness of our QDP in solving WQMR problems. Additionally, on larger systems (e.g., 53-qubit Rochester), we observe that both WCF and AVF may reduce for QDP and all baselines.

In Table II, δ represents the average number of two-qubit gates per logical qubit. In general, a higher number of two-qubit gates indicates stronger quantum entanglement within the circuit [52]. QDP achieves greater WCF improvement on circuits with stronger quantum entanglement. For example, on circuit dist_223 ($\delta = 4.69$), QDP improves WCF by 8.08% over SATMAP. On circuit rd53_138, which has a higher δ of 7.50, QDP achieves larger improvements of 9.88%. Although both circuits have the same number of gates, QDP performs better on the one with stronger entanglement.

The WCF improvement by QDP is especially remarkable when solving deep quantum circuits ($>$ 100 gates). For example, on circuit 9symml_195, which contains 121 gates, QDP improves up to 26.9% WCF compared with the baselines. This is because the baselines repeatedly use the same limited subset of physical qubits. In contrast, QDP distributes gates more evenly across a larger set of physical qubits, avoiding the overuse of any single qubit. For example, when solving circuit 9symml_195, QMAP uses only 12 physical qubits, while QDP uses 17.

B. Runtime Comparison

Table III compares the runtime of QDP with QMAP, SABRE, and SATMAP. QDP significantly outperforms SATMAP in runtime. On average, QDP is $125.85\times$ faster (4.247 s vs. 534.50 s) due to the efficiency of our multi-threaded DP algorithm over SAT-based exhaustive search. On the other hand, SABRE and QMAP are faster than QDP because their greedy heuristics explore a more limited search space compared to our DP. However, since QMR is typically executed only once in practice, the resulting mapping and routing remain fixed for all subsequent executions. Therefore, it is worthwhile to invest more time in finding a reliable QMR solution, as it directly impacts the long-term fidelity of quantum computation.

C. Solution Quality under Different Architecture Connectivity

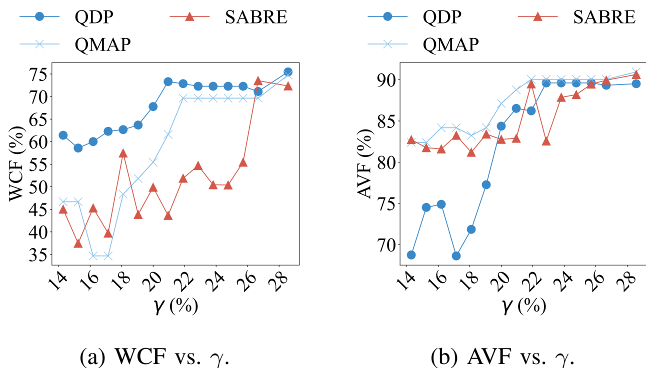
As architecture connectivity plays the key role in the solution quality of a QMR algorithm, we study how different

Authorized licensed use limited to: University of Wisconsin. Downloaded on May 16,2026 at 20:57:39 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2026.3693628

TABLE III: Comparison of runtime (s) among QDP, QMAP, SABRE, and SATMAP on the 20 QMR benchmarks.

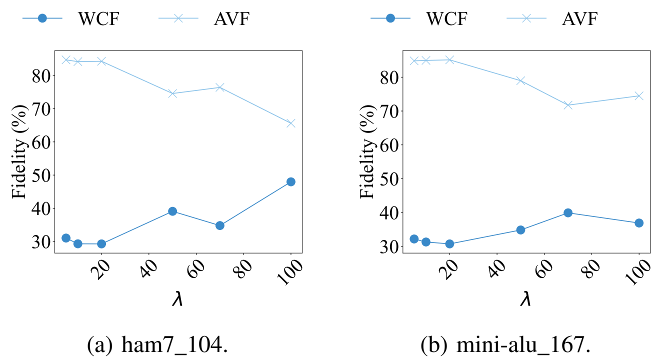
Arch.	Circuits	Runtime (s)			
Name	Name	QDP	QMAP	SABRE	SATMAP
Mel-bourne	rd32-v0_66	0.031	0.031	0.020	2.349
	4gt13_90	0.331	0.038	0.059	122.460
	ex1_226	0.001	0.029	0.009	0.772
	alu-bdd_288	0.365	0.037	0.045	12.162
	rd53_138	2.66	0.041	0.074	148.376
Tokyo	mini_alu_305	8.777	0.044	0.088	2,098.201
	9symml_195	4.871	0.041	0.071	39.036
	dist_223	5.052	0.043	0.073	2,552.787
	0410184_169	13.809	0.040	0.063	1,146.441
	Combo	4.557	0.040	0.062	67.414
Rochester	ham7_104	7.255	0.068	0.273	> 2h
	mini-alu_167	6.069	0.069	0.163	> 2h
	mod5adder_127	22.417	0.09	0.333	–
	mod10_171	3.773	0.0585	0.137	2784.48
	sf_276	41.491	0.145	0.428	7009.42
	rd53_131	23.232	0.0854	0.284	> 2h
	rd53_311	117.716	0.065	0.204	–
	sym6_316	65.607	0.066	0.195	> 2h
	mod8-10_178	10.07	0.077	0.196	5184.67
	wim_266	96.366	0.163	0.612	> 2h
Average (geomean)		4.247	0.056	0.169	534.50



(a) WCF vs. γ. (b) AVF vs. γ.

Fig. 11: Comparison of WCF and AVF across QDP, QMAP, and SABRE for different architecture connectivity (γ).

connectivity affects our QMR results. We denote architecture connectivity, γ, as the graph density [53] of the architecture coupling graph. Figure 11 compares the WCF and AVF across QDP, QMAP, and SABRE for different values of γ. To sample various γ values, we insert and remove edges in the 15-qubit Melbourne architecture. We then run circuit rd53_138 on each architecture. As shown in Figure 11a, a higher γ generally leads to a higher WCF for all QMR algorithms. For example, when γ is 14.3%, the WCF values of QDP, QMAP, and SABRE are 61.42%, 46.7%, and 45.0%, respectively. At $\gamma = 28.6\%$, the WCF values increase to 75.5%, 74.8%, and 72.4%. AVF follows a similar increasing trend (Figure 11b), For example, at $\gamma = 14.3\%$, the AVFs of QDP, QMAP, and SABRE are 68.75%, 82.3%, and 82.7%, respectively. As γ increases to 28.6%, the AVF of the three algorithms converges to around 90%. This is because higher γ reduces the number of required `swap` gates and increases routing flexibility, improving both AVF and WCF. Despite similar AVF levels at high γ, QDP consistently achieves higher WCF than QMAP and SABRE.



(a) ham7_104. (b) mini-alu_167.

Fig. 12: WCF and AVF under different λ across two QMR benchmarks.

D. Solution Quality under Different λ

Figure 12 shows the solution quality under different values of λ on two QMR benchmarks. When λ increases, WCF increases while AVF decreases. As shown in Figure 12a, at $\lambda = 5$, the WCF and AVF values are 31.0% and 84.7%, respectively. When λ increases to 100, these values change to 47.97% and 65.6%. This is because λ acts as a trade-off parameter between WCF and AVF in the path-finding cost (Equation 5). A larger λ leads to a more balanced gate usage across physical qubits but requires more gates, prioritizing WCF over AVF. Likewise, Figure 12b follows a similar trend. The best selection strategy of λ depends on the circuit and quantum computer architecture. Since there is no universal optimal value, we parameterize it for applications.

E. Solution Quality under Different k

Figure 13 shows the solution quality under different values of k on two QMR benchmarks. In general, increasing k retains more mappings and reduces pruning, which slightly increases WCF but can mildly decrease AVF due to QDP's WCF-oriented selection. As shown in Figure 13a, the WCF and AVF values are 71.92% and 82.19%, respectively, at $k = 50$. When k increases to 1000, these values change to 74.72% and 80.83%. In contrast, Figure 13b shows a much flatter trend: WCF and AVF change only marginally as k increases. This is because QDP prioritizes the highest-WCF candidates in each state, and a small k already preserves the most influential mappings. Increasing k mainly adds lower-ranked candidates that rarely change the final WCF.

Figure 14 shows the runtime of QDP for different k on the same benchmarks. The runtime scales approximately linearly with k, which is expected because a larger k increases the number of mapping candidates processed per state (Section III-B), introducing additional work roughly proportional to k.

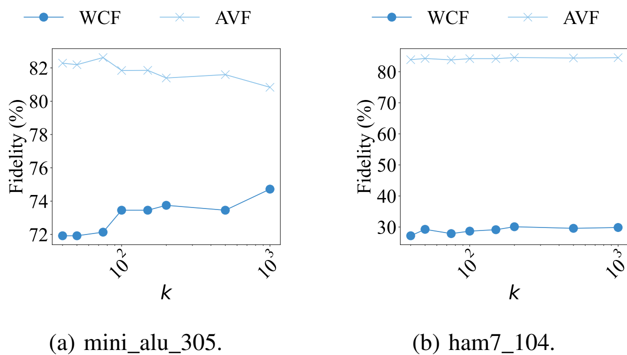
F. Bottlenecks on QDP Parallelism

We study two dominant bottlenecks that limit the parallel performance of QDP:

- **Algorithmic parallelism:** QDP performs a level-synchronous traversal over a bucket list (Algorithm 1).

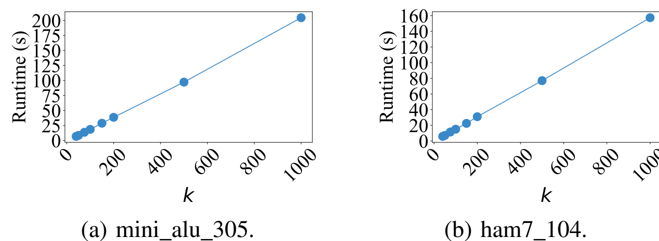
Authorized licensed use limited to: University of Wisconsin. Downloaded on May 16,2026 at 20:57:39 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2026.3693628



(a) mini_alu_305. (b) ham7_104.

Fig. 13: WCF and AVF under different k across two QMR benchmarks.



(a) mini_alu_305. (b) ham7_104.

Fig. 14: QDP Runtime under different k across two QMR benchmarks.

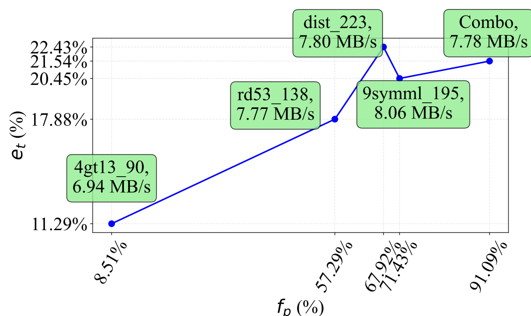
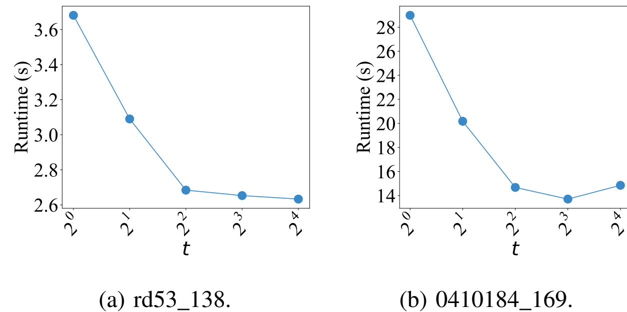


Fig. 15: e_t of QDP across circuits with varying f_p and shared-data throughput (MB/s). Annotations show the name of each circuit and the corresponding shared-data throughput.

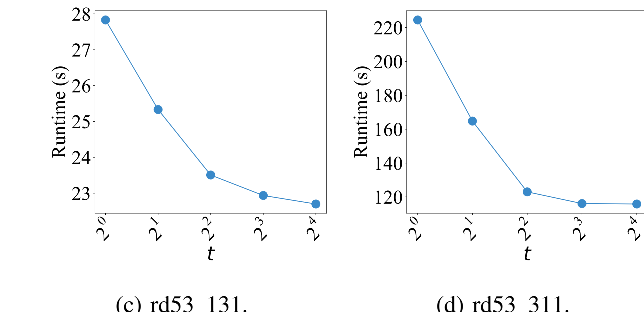
Only buckets containing multiple states can be processed by multiple threads. We therefore define the *parallelizable fraction* (f_p) as the number of multi-state (parallelizable) buckets divided by the total number of buckets. QDP is bounded by this inherent algorithmic parallelism.

- **Shared-data contention:** Within parallelizable buckets, multiple threads repeatedly access shared data structures (e.g., B and D), which introduces serialization and contention on a shared-memory machine. We quantify this bottleneck using *shared-data throughput* (MB/s), defined as the estimated total bytes read/written to these shared structures divided by QDP wall-clock time.

To decouple algorithmic parallelism from shared-data contention, we select five circuits (dist_223, 9symml_195, 4gt13_90, rd53_138, and combo) with similar shared-data throughput (6.9–8.0 MB/s). This allows us to examine how



(a) rd53_138. (b) 0410184_169.



(c) rd53_131. (d) rd53_311.

Fig. 16: Runtime (s) of QDP under different number of threads (t) across four QMR benchmarks.

parallel efficiency varies with f_p under comparable shared-memory contention. Specifically, we report *thread efficiency* (e_t), defined as the wall-clock speed-up achieved with eight threads normalized by eight [54]. e_t is largely workload-agnostic and fairly reflects how effectively QDP converts additional threads into speed-up. As shown in Figure 15, e_t increases with f_p until f_p reaches about 60%. Beyond this point, e_t saturates, indicating that QDP transitions from being primarily parallelism-bound to being primarily memory-bound. Consistently, 9symml_195 achieves lower e_t than dist_223 (20.4% vs. 22.4%) despite a higher f_p (71.43% vs. 67.92%), which aligns with its higher shared-data throughput (8.06 vs. 7.8 MB/s) and suggests more time spent on contended shared-data accesses.

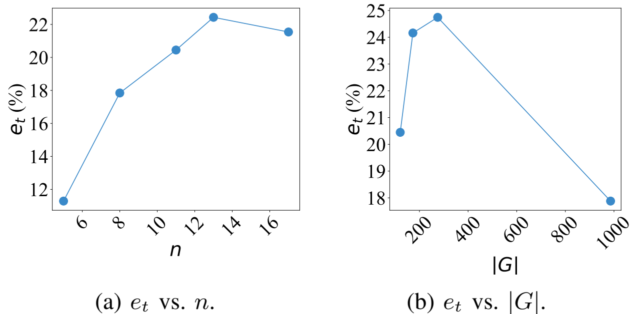
G. Parallel Scalability of QDP

1) Scalability of QDP under Increasing Number of Threads: Figure 16 illustrates the runtime of QDP at different numbers of threads on four QMR benchmarks. When the number of threads increases, QDP can complete QMR faster. For instance, at eight threads, QDP is $1.93\times$ faster than one thread on rd53_311. QDP's performance benefits from multi-threading saturate at around 16 threads. This is due to shared-data contention (Section IV-F), which introduces overhead that eventually outweighs the benefits of multi-threading.

2) Scalability of QDP under Increasing Circuit Size: To study scalability under increasing circuit size, we plot thread efficiency (e_t, Section IV-F) vs. the number of qubits (n) (Figure 17a) and vs. the number of gates ($|G|$) (Figure 17b). To ensure a fair comparison, we run the e_t vs. n experiment

Authorized licensed use limited to: University of Wisconsin. Downloaded on May 16,2026 at 20:57:39 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2026.3693628



(a) e_t vs. n.

(b) e_t vs. $|G|$.

Fig. 17: Thread efficiency (e_t) of QDP under different circuit sizes (i.e., n and $|G|$).

on 4gt13_90, rd53_138, 9symml_195, dist_223, and Combo, which have similar $|G|$ ranging from 107 to 132. Similarly, we run the e_t vs. $|G|$ experiment on mini_alu_305, 9symml_195, rd53_311, and wim_266, which have similar n (10 to 13).

In Figure 17a, e_t increases with n initially but saturates at $n = 17$ (Combo). This is because QDP is parallelism-bound initially: e_t is 11.29% on 4gt13_90 ($n = 5$) when f_p (Section IV-F) is only 8.51%, but increases to 22.43% on dist_223 ($n = 13$) as f_p rises to 67.92%. For circuits with similar gate counts, a larger n can increase available parallelism (higher f_p) by enabling more possible state expansions. As f_p grows further, QDP becomes increasingly memory-bound and performance saturates due to contention on shared data. For example, e_t drops to 21.54% on 17-qubit Combo when f_p increases to 91.09%.

In Figure 17b, there is no clear monotonic trend in e_t with $|G|$ because circuits with similar n can have very different f_p depending on their circuit structure. For example, rd53_311 ($|G| = 275$) has $f_p = 87.65\%$, while wim_266 ($|G| = 986$) has $f_p = 44.43\%$. When f_p is low, QDP is parallelism-bound, indicating scalability with respect to $|G|$ depends primarily on the circuit structure rather than on $|G|$ alone.

H. Comparison with an Adapted AQMR Algorithm

To further justify the need for a dedicated algorithm for WQMR rather than directly adapting an AQMR solution, we compare QDP with a baseline that augments an AQMR algorithm with WCF post-selection.

In this baseline, for each $i \in \{n, \ldots, N\}$, we iteratively execute an AQMR algorithm while restricting the available physical qubits to the subset $\{p_0, \ldots, p_{i-1}\}$. This process generates multiple candidate solutions. Among all valid candidates, we select the one with the highest WCF as the final solution (i.e., post-selection).

We implement this baseline using QMAP as the AVF-centric backend. QMAP is chosen because it is deterministic and therefore ensures reproducible comparisons. We evaluate six circuits across the three IBM architectures (Melbourne, Tokyo, and Rochester).

Table IV reports the results. Post-selection leaves WCF unchanged in three cases (0410184_169, mod8-10_178, and wim_266) and yields only marginal improvements in the remaining three cases (alu-bdd_288, rd53_138, and dist_223). The gains are limited because the underlying AQMR algorithm still optimizes AVF. Post-selection can only improve results when QMAP happens to produce a higher WCF candidate for some i. Moreover, improvements are more likely when the architecture provides multiple routing alternatives, whereas on sparser topologies, such as Rochester, the flexibility is limited and WCF often remains unchanged.

TABLE IV: WCF comparison of QMAP, QMAP with post-selection (QMAP-PS), and QDP. The table also reports the WCF improvement achieved by post-selection (Improv. from PS).

Arch.	Circuits	QMAP	QMAP-PS	Improv. from PS	QDP
Melbourne	alu-bdd_288	48.94%	52.40%	3.46%	74.42%
	rd53_138	51.83%	56.22%	4.39%	61.24%
Tokyo	dist_223	59.78%	63.56%	3.78%	76.31%
	0410184_169	68.88%	68.88%	0	77.78%
Rochester	mod8-10_178	17.25%	17.25%	0	27.82%
	wim_266	1.84%	1.84%	0	9.51%

In contrast, QDP consistently achieves substantially higher WCF on the same circuits. For example, on alu-bdd_288, post-selection improves WCF from 48.94% to 52.40%, whereas QDP achieves 74.42% WCF. This result suggests that simply wrapping an AQMR algorithm with WCF post-selection is insufficient.

V. Limitation and Future Work

While QDP demonstrates significant improvements in WCF across various quantum circuits and architectures, several limitations remain, opening opportunities for future research.

A. Generalization to Drifting Fidelities

QDP currently assumes fidelity is constant with respect to depth. While modeling depth-dependent drift is out of scope for this paper, we believe QDP can be extended in the future to consider such models into our DP cost models, because QDP maximizes WCF while balancing AVF at each step of the circuit through its DP formulation.

B. Generalization to Fault-tolerant Architectures

Although QDP is designed for fidelities and architectures in the noisy intermediate-scale quantum (NISQ) era, improving worst-case behavior in QMR is a general objective that extends beyond NISQ systems. We anticipate that QDP can be adapted to fault tolerant (FT) circuits and surface-code based architectures by replacing the NISQ cost model based on gate fidelities with an FT-aware cost model.

For example, in surface code architectures, a logical qubit occupies a 2D patch whose physical qubit footprint scales with the patch area, that is, $O(d^2)$ in the code distance d [55]. An FT-aware cost model can therefore be expressed directly in terms of d. QDP can reduce the required worst-case code distance. Since physical qubit overhead scales as $O(d^2)$, even a modest reduction in the worst-case d can yield a superlinear reduction in total resource overhead, which may not be captured by AVF-centric objectives. A potential

Authorized licensed use limited to: University of Wisconsin. Downloaded on May 16,2026 at 20:57:39 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2026.3693628

challenge is that applying QDP in FT settings may require additional pruning strategies to maintain tractability as circuit size becomes increasingly large.

VI. Conclusion

In this paper, we have formulated a WCF-aware QMR problem and introduced a parallel DP-based QMR algorithm called QDP that optimizes the WCF of physical qubits. We have evaluated QDP on commonly used QMR benchmarks three representative IBM quantum computer architectures with up to 53 qubits. Compared with state-of-the-art QMR algorithms, QDP can enhance the WCF by up to 33.88% while achieving comparable AVF.

Acknowledgment

The work described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project Reference Number: T46-415/25-R). This work was conducted in the JC STEM Lab of Intelligent Design Automation funded by The Hong Kong Jockey Club Charities Trust. This work was also supported by NSF grants 2235276, 2349144, 2349143, 2349582, and 2349141. We acknowledge the use of generative AI tools to assist with language polishing of the manuscript and the response letter, as well as with code development. Specifically, GPT-5.4, GPT-5.3-Codex, and GPT-4.1 were used.

References

[1] 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.

[2] M. Y. Siraichi, V. F. d. Santos, C. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, 2018, pp. 113–125.

[3] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the ibm qx architectures," *IEEE TCAD*, vol. 38, no. 7, pp. 1226–1236, 2018.

[4] C. Y. Huang and W. K. Mak, "Efficient qubit routing using a dynamically-extract-and-route framework," *IEEE TCAD*, 2024.

[5] A. Sinha, U. Azad, and H. Singh, "Qubit routing using graph neural network aided monte carlo tree search," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 36, no. 9, 2022, pp. 9935–9943.

[6] A. Molavi, A. Xu, M. Diges, L. Pick, S. Tannu, and A. Albarghouthi, "Qubit mapping and routing via maxsat," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1078–1091.

[7] R. Wille, L. Burgholzer, and A. Zulehner, "Mapping quantum circuits to ibm qx architectures using the minimal number of swap and h operations," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[8] I. Shaik and J. van de Pol, "Optimal layout synthesis for quantum circuits as classical planning," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[9] R. Wille, O. Keszocze, M. Walter, P. Rohrs, A. Chattopadhyay, and R. Drechsler, "Look-ahead schemes for nearest neighbor optimization of 1d and 2d quantum circuits," in *2016 21st Asia and South Pacific design automation conference (ASP-DAC)*. IEEE, 2016, pp. 292–297.

[10] 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.

[11] 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.

[12] L. Liu and X. Dou, "Qucloud: A new qubit mapping mechanism for multi-programming quantum computing in cloud environment," in *2021 IEEE HPCA*. IEEE, 2021, pp. 167–178.

[13] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers," in *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, 2019, pp. 1015–1029.

[14] A. Ash-Saki, M. Alam, and S. Ghosh, "Qure: Qubit re-allocation in noisy intermediate-scale quantum computers," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[15] J. Liu, E. Younis, M. Weiden, P. Hovland, J. Kubiatowicz, and C. Iancu, "Tackling the qubit mapping problem with permutation-aware synthesis," *Proceedings - 2023 IEEE International Conference on Quantum Computing and Engineering, QCE 2023*, vol. 1, pp. 745–756, 2023.

[16] P. Zhu, Z. Guan, and X. Cheng, "A dynamic look-ahead heuristic for the qubit mapping problem of nisq computers," *IEEE TCAD*, vol. 39, pp. 4721–4735, 12 2020.

[17] D. Bhattacharjee, A. A. Saki, M. Alam, A. Chattopadhyay, and S. Ghosh, "Muqut: Multi-constraint quantum circuit mapping on nisq computers," in *2019 IEEE/ACM international conference on computer-aided design (ICCAD)*. IEEE, 2019, pp. 1–7.

[18] H. Wang, P. Liu, D. B. Tan, Y. Liu, J. Gu, D. Z. Pan, J. Cong, U. A. Acar, and S. Han, "Atomique: A quantum compiler for reconfigurable neutral atom arrays," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 293–309.

[19] S. Khandavilli, I. Palanisamy, M. V. Nguyen, T. V. Le, T. N. Nguyen, and T. N. Dinh, "Towards fidelity-optimal qubit mapping on nisq computers," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 01, 2023, pp. 89–98.

[20] B. Tan, D. Bluvstein, M. D. Lukin, and J. Cong, "Qubit mapping for reconfigurable atom arrays," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3508352.3549331

[21] C.-Y. Huang and W.-K. Mak, "Ctqr: Control and timing-aware qubit routing," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024, pp. 140–145.

[22] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: A case for variability-aware policies for nisq-era quantum computers," in *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, 2019, pp. 987–999.

[23] P. Zhu, W. Ding, L. Wei, X. Cheng, Z. Guan, and S. Feng, "A variation-aware quantum circuit mapping approach based on multi-agent cooperation," *IEEE Transactions on Computers*, vol. 72, no. 8, pp. 2237–2249, 2023.

[24] S. Niu, A. Suau, G. Staffelbach, and A. Todri-Sanial, "A hardware-aware heuristic for the qubit mapping problem in the nisq era," *IEEE Transactions on Quantum Engineering*, vol. 1, pp. 1–14, 2020.

[25] N. Paraskevopoulos, F. Sebastiano, C. G. Almudever, and S. Feld, "Spinq: Compilation strategies for scalable spin-qubit architectures," *ACM Transactions on Quantum Computing*, vol. 5, no. 1, Dec. 2023. [Online]. Available: https://doi.org/10.1145/3624484

[26] M. A. Steinberg, S. Feld, C. G. Almudever, M. Marthaler, and J.-M. Reiner, "Topological-graph dependencies and scaling properties of a heuristic qubit-assignment algorithm," *IEEE Transactions on Quantum Engineering*, vol. 3, pp. 1–14, 2022.

[27] S.-T. Huang, Y.-J. Jiang, S.-Y. Fang, and C.-K. Cheng, "Smt-based layout synthesis for silicon-based quantum computing with crossbar architecture," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '24. New York, NY, USA: Association for Computing Machinery, 2025. [Online]. Available: https://doi.org/10.1145/3676536.3676819

[28] Y. Zhu, Y. Zhou, J. Cheng, Y. Jin, B. Li, S. Niu, and Z. Liang, "Compiler optimizations for qaoa," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '24. New York, NY, USA: Association for Computing Machinery, 2025. [Online]. Available: https://doi.org/10.1145/3676536.3697127

[29] R. Wille and L. Burgholzer, "Mqt qmap: Efficient quantum circuit mapping," in *Proceedings of the 2023 International Symposium on Physical Design*, 2023, pp. 198–204.

[30] X. Ren, T. Zhang, X. Xu, Y.-C. Zheng, and S. Zhang, "Invited: Leveraging machine learning for quantum compilation optimization," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3663510

Authorized licensed use limited to: University of Wisconsin. Downloaded on May 16,2026 at 20:57:39 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edit
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2026.3693628

[31] P. Escofet, A. Gonzalvo, E. Alarcón, C. G. Almudéver, and S. Abadal, "Route-forcing: Scalable quantum circuit mapping for scalable quantum computing architectures," in *2024 IEEE QCE*, vol. 01, 2024, pp. 909–920.

[32] H. Kim, E. Jang, S. Choi, Y. Kim, and W. W. Ro, "Qr-map: A map-based approach to quantum circuit abstraction for qubit reuse optimization," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ser. ISCA '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1568–1582. [Online]. Available: https://doi.org/10.1145/3695053.3731020

[33] L. Lao, H. van Someren, I. Ashraf, and C. G. Almudever, "Timing and resource-aware mapping of quantum circuits to superconducting processors," *IEEE TCAD*, vol. 41, no. 2, pp. 359–371, 2022.

[34] F. J. Cardama, J. Vázquez-Pérez, T. F. Pena, J. C. Pichel, and A. Gómez, "Quantum compilation process: A survey," in *European Conference on Parallel Processing*. Springer, 2024, pp. 100–112.

[35] D. Rattacaso, D. Jaschke, M. Ballarin, I. Siloi, and S. Montangero, "Quantum circuit compilation with quantum computers," *Physical Review Research*, vol. 7, no. 3, p. 033268, 2025.

[36] D. Venturelli, M. Do, B. O'Gorman, J. Frank, E. Rieffel, K. E. Booth, T. Nguyen, P. Narayan, and S. Nanda, "Quantum circuit compilation: An emerging application for automated reasoning," in *Scheduling and Planning Applications Workshop*, 2019.

[37] J. Yang, Y. A. Kharkov, Y. Shi, M. J. Heule, and B. Dutertre, "Quantum circuit mapping based on incremental and parallel sat solving," in *27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024, pp. 29–1.

[38] P. Liu, J. Arora, M. Xu, and U. A. Acar, "Popqc: Parallel optimization for quantum circuits," in *Proceedings of the 37th ACM Symposium on Parallelism in Algorithms and Architectures*, 2025, pp. 269–283.

[39] A. Botea, A. Kishimoto, and R. Marinescu, "On the complexity of quantum circuit compilation," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 9, no. 1, 2018, pp. 138–142.

[40] G. Acampora and R. Schiattarella, "Deep neural networks for quantum circuit mapping," *Neural Computing and Applications*, vol. 33, no. 20, pp. 13 723–13 743, 2021.

[41] P. Zhu, S. Zheng, L. Wei, X. Cheng, Z. Guan, and S. Feng, "The complexity of quantum circuit mapping with fixed parameters," *Quantum Information Processing*, vol. 21, no. 10, p. 361, 2022.

[42] L. Burgholzer, S. Schneider, and R. Wille, "Limiting the search space in optimal quantum circuit mapping," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022, pp. 466–471.

[43] H. Fan, C. Guo, and W. Luk, "Optimizing quantum circuit placement via machine learning," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 19–24.

[44] R. Bellman, "Dynamic programming treatment of the travelling salesman problem," *Journal of the ACM (JACM)*, vol. 9, no. 1, pp. 61–63, 1962.

[45] Y. Zhao, Y. Guo, Y. Yao, A. Dumi, D. M. Mulvey, S. Upadhyay, Y. Zhang, K. D. Jordan, J. Yang, and X. Tang, "Q-gpu: A recipe of optimizations for quantum circuit simulation using gpus," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 726–740.

[46] E. Kjelkerud and O. Thessén, "Generation of hazard free tests using the d-algorithm in a timing accurate system for logic and deductive fault simulation," in *Proceedings of the 16th Design Automation Conference*, ser. DAC '79. IEEE Press, 1979, p. 180–184.

[47] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," *IBM journal of Research and Development*, vol. 10, no. 4, pp. 278–291, 1966.

[48] R. H. B. Netzer and B. P. Miller, "What are race conditions? some issues and formalizations," *ACM Lett. Program. Lang. Syst.*, vol. 1, no. 1, p. 74–88, Mar. 1992. [Online]. Available: https://doi.org/10.1145/130616.130623

[49] "Openmp critical." [Online]. Available: https://www.ibm.com/docs/en/zos/2.4.0?topic=processing-pragma-omp-critical

[50] E. W. Dijkstra, "A note on two problems in connexion with graphs," in *Edsger Wybe Dijkstra: his life, work, and legacy*, 2022, pp. 287–290.

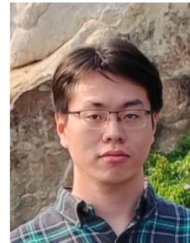
[51] "Ibm quantum computers." [Online]. Available: https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/ibm-backend

[52] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[53] "Graph density." [Online]. Available: https://en.wikipedia.org/wiki/Dense_graph

[54] P. Pacheco, *An introduction to parallel programming*. Elsevier, 2011.

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



Shui Jiang received his BS degree in electrical engineering from Zhejiang University in 2022. He is currently studying for his PhD degree in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research interests include high-performance computing and quantum computing.

Wen Cheng received the B.S. and M.S. degrees in Computer Science from National Tsing Hua University (NTHU). He conducted his research in the Theta Lab under the supervision of Prof. Tsung-Yi Ho, focusing on computer-aided design methodologies for qubit routing and optimization of quantum circuit compilation.

Yi-Hua Chung is a third-year Ph.D. student in the Department of Electrical and Computer Engineering at the University of Wisconsin–Madison, where she is advised by Prof. Tsung-Wei Huang. Her research focuses on high-performance computing and electronic design automation, with an emphasis on GPU-accelerated logic simulation and gate sizing for large-scale circuits.

Tsung-Yi Ho is a Professor in the Department of Computer Science and Engineering, the Chinese University of Hong Kong (CUHK). He received his Ph.D. in Electrical Engineering from National Taiwan University in 2005. His research interests include several areas of computing and emerging technologies, especially in design automation of microfluidic biochips. He is a Distinguished Member of ACM and a Fellow of IEEE.

Tsung-Wei Huang is an Associate Professor in ECE at the University of Wisconsin-Madison, with an affiliate appointment in CS. Previously, he was an Assistant Professor at Utah (2019–2023). He earned his PhD in ECE from UIUC (2017) and BS/MS in CS from Taiwan's NCKU (2011). His research focuses on software systems for performance-critical applications, including CAD, machine learning, and quantum computing.

Authorized licensed use limited to: University of Wisconsin. Downloaded on May 16,2026 at 20:57:39 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.