Tackling the Qubit Mapping Problem with Permutation-Aware Synthesis

Ji Liu[*,†], Ed Younis[‡,†], Mathias Weiden[§], Paul Hovland[*], John Kubiatowicz[§], Costin Iancu[‡]
[*] Mathematics and Computer Science Division, Argonne National Laboratory
[‡] Computational Research Division, Lawrence Berkeley National Laboratory
[§]Department of Electrical Engineering and Computer Science, University of California, Berkeley
[†] Contributed equally to this work

2023 IEEE International Conference on Quantum Computing and Engineering (QCE) | 979-8-3503-4323-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/QCE57702.2023.00090

***Abstract*— We propose a scalable, hierarchical qubit mapping and routing algorithm that harnesses the power of circuit synthesis. First, we decompose large circuits into subcircuits small enough to be directly resynthesized. For each block, we pre-synthesize them for all permutations of its input and output qubits. Following this offline step, we employ a permutation-aware, block-based generalization of the popular SABRE mapping algorithm. This mapping step stitches together blocks by choosing an input-output permutation that minimizes intra-block gate count and required inter-block communication (SWAP and bridge gates). Our approach has a twofold advantage: 1) circuit synthesis may eliminate more two-qubit gates than other optimizing compilers; 2) considering all permutations of input and output qubits eliminates communication operations transparently. In contrast, other mapping algorithms can only introduce communication operations. We show that we can produce better-quality circuits than commercial compilers: shorter by up to 68% (18% on average) fewer gates than Qiskit, up to 36% (9% on average) fewer gates than Tket. We outperform BQSkit, a permutation-unaware, synthesis-based compiler, by up to 67% (21% on average) fewer gates. We also exceed experimental optimal mappers such as OLSQ in quality (10.7% shorter circuits) and time to solution. Our scalable, heuristic approach can be seamlessly integrated into any quantum circuit compiler or optimization infrastructure, and it applies well to any qubit technology, such as superconducting and trapped ions.**

I. Introduction

Circuit depth and gate count are direct measures of quantum program performance in the circuit model [28]. Accordingly, compilation infrastructures, such as Qiskit, TKET, and BQSkit [7], [37], [47], minimize these using a variety of approaches. Hardware-agnostic optimizations first delete redundant gates by using functional equivalence [12], [21], [42], pattern rewriting [16], [37] or circuit resynthesis [45], [46] techniques. Qualitatively, synthesis-based optimizations provide higher gate count reduction than peephole or pattern rewriting methods; however, complete compilation pipelines combine all methods. For small circuits, synthesis starts with the unitary representation of a program and discovers a new shorter implementation [6], [32], [38], [48]. This process is a form of global optimization. For large circuits it employs [30], [45] a divide-and-conquer approach based on partitioning and direct synthesis of small circuits.

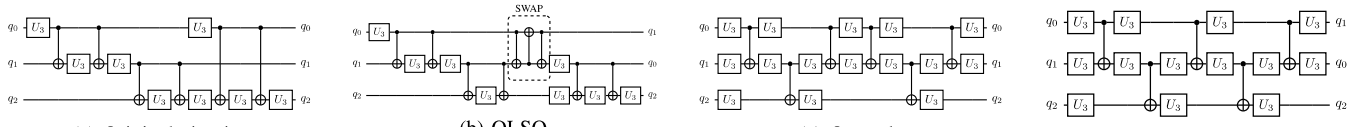
Domain scientists usually develop circuits without consideration of hardware connectivity constraints, leaving this problem to be tackled using mapping and routing algorithms [17], [20], [22], [24], [27], [39]. While "optimizations" delete gates, mapping introduces additional gates to perform communication (SWAP) between qubits that are not directly connected. This qubit mapping problem is known to be NP-hard [4]. Most existing algorithms consider only a pair of qubits as endpoints at any given time and introduce two-qubit entangling gates (e.g., CNOT, iSWAP) between these.

This paper presents a novel circuit mapping approach that integrates the power of topology-aware synthesis into traditional mapping algorithms. A large circuit is first partitioned into smaller blocks that are directly synthesized to maximize gate count reduction. For each block, we synthesize circuits for all mapping of input qubit permutations to output qubit permutations. This consideration ensures we can find the shortest possible implementation, and we refer to this as permutation-aware synthesis (PAS). Each input-output permutation of a $k-qubit$ block is also synthesized for all the $k-qubit$ couplings embedded into the physical hardware connectivity. This ensures that we apriori find the best possible mapping of the best possible implementation: we do not introduce spurious qubit communication within this step, and we can exploit hardware connectivity richer than the logical connectivity. For mapping, we extend the SABRE [20] algorithm to consider interactions between permutations of many-qubit blocks. SABRE divides the circuit into multiple layers and iteratively routes the gates in the front layer. It selects the best route based on a heuristic cost function considering the distance between mapped physical qubits. We refer to this entire process as permutation-aware mapping (PAM).

The central intuition behind our approach is that by extending traditional mapping algorithms to consider many-qubit block interactions and all their permutations (PAS), we can make a selection (PAM) that *minimizes gate count within each block together with the required inter-block communication*, given the hardware connectivity constraints. This work makes the following contributions:

- We introduce the idea of permutation awareness and propose Permutation-Aware Synthesis (PAS). Considering arbitrary input-output qubit permutations at the unitary level leads to shorter circuits, and it finds the permutation that minimizes routing cost. This is described in Section IV.
- We present Permutation-Aware Mapping (PAM), a block-

979-8-3503-4323-6/23/$31.00 ©2023 IEEE
DOI 10.1109/QCE57702.2023.00090



(a) Original circuit (b) OLSQ (c) Qsearch (d) PAS

Fig. 1: 3-qubit quantum Fourier transform (QFT) mapped to a linear topology with different algorithms

based qubit mapping framework. PAM employs PAS's optimization and mapping potential together with block-level routing heuristics. This is described in Section V.

- We demonstrate the ability to leverage rich hardware connectivity directly. On these architectures, the logical circuit connectivity is often directly embedded in the hardware connectivity; that is, the circuit is already mapped, and existing algorithms will not modify it. In contrast, our topology-aware PAS step can entirely restructure circuits to exploit the hardware. This particularly benefits fully-connected architectures, such as trapped ion qubits, as discussed in Section VIII.

We have implemented our algorithm within the BQSKit compilation infrastructure and evaluated it on a series of benchmarks previously used for assessing mapping algorithms and across multiple existing and proposed future architectures, up to 1024 qubits. We show that we can produce better-quality circuits than commercial compilers and their state-of-the-art mappers: shorter by up to 68% (18% on average) fewer gates than Qiskit, up to 36% (9% on average) fewer gates than Tket. We outperform BQSkit, a permutation-unaware, synthesis-based compiler, by up to 67% (21% on average) fewer gates. We also exceed experimental optimal mappers such as OLSQ in quality (10.7% shorter circuits) and time to solution. Our scalable, heuristic approach can be seamlessly integrated into any quantum circuit compiler or optimization infrastructure, and it applies well to any qubit technology, such as superconducting and trapped ions.

II. Background and Motivation

Quantum compilers must produce circuits containing multi-qubit gates only between physically connected qubits. This process can be decomposed into two steps: finding the initial logical-to-physical qubit pairing (mapping) and applying SWAP gates to move the qubits to physically connected qubits (routing). Qubit mapping and routing is NP-hard [4], and previous algorithms are classified into two categories: heuristic or optimal mapping algorithms.

A. Heuristic Mappers

SABRE [20] is a canonical heuristic algorithm that has been adopted by the Qiskit and BQSKit compilers [7], [47], as well as multiple routing algorithms [22], [29]. SABRE first divides the circuit into layers. The algorithm then routes gates in the front layer and selects a path using a heuristic cost function based on the distance between mapped physical qubits. The heuristic cost function routes the front layer with lookahead. It balances the routing cost for the gates in the front layer and the gates in the extended layer, comprised of gates that will be routed in the future. The initial mapping is updated based on the reverse traversal of the circuit.

SABRE has inspired several heuristic algorithms. Liu et al. [22] proposed an optimization-aware heuristic that minimizes the number of 2-qubit gates after circuit optimizations. Niu et al. proposed a layered hardware-aware heuristic [29] based on calibration data. Other heuristic algorithms include TKET [37], commutation-based routing [17], simulated annealing-based routing [50], dynamic lookahead [51], and time-optimal mapping [49].

B. Optimal Mapppers

Optimal mappers convert the problem into constraints and find the circuit with optimal SWAP gate count or depth using optimal solvers. For example, the OLSQ [39] approach formulates mapping and routing as a satisfiability modulo theory (SMT) optimization problem and then uses the Z3 SMT solver [26] to find the optimal circuit. The BIP mapper [27] in Qiskit finds the optimal mapping and routing by solving a binary integer programming (BIP) problem. Because of the exponential growth of the search space, constraint-based solvers usually face scalability issues.

C. Synthesis for Mapping

A unitary synthesis algorithm generates a quantum circuit starting from a unitary matrix representation of the input. Albeit limited to handling small problems, good direct synthesis algorithms usually produce better quality circuits than pattern rewriting-based quantum compilers alone. The BQSKit [47] compilation framework can handle very large circuits using a combination of algorithms to partition circuits into smaller subcircuits (blocks) and direct synthesis methods for each block.

In this paper, we make use of the capability of topology-aware direct synthesis as illustrated by the QSearch [6] algorithm. Qsearch employs an A* heuristic to search over a tree of possible circuits based on device topology: it builds circuits bottom-up, and at each step, it attempts to place a CNOT gate only between physically connected qubits. This approach enables it to generate near-optimal depth circuits for any physical qubit connectivity.

Synthesis algorithms directly construct a circuit based on the unitary matrix, regardless of the original circuit structure. In particular, when comparing against mapping algorithms, the

latter has to maintain the circuit structure and can only introduce communication (SWAPs), while QSearch can potentially delete redundant communication present in the original circuit.

The following 3-qubit QFT algorithm example illustrates this optimization potential. The best-known implementation of this algorithm [43] is shown in Figure 1a, which contains six CNOT gates and assumes all-to-all connectivity. As shown in Figure 1b, mapping the circuit onto a linear topology with the optimal routing algorithm OLSQ [39] adds a single SWAP gate to make a total of nine CNOTs. As shown in Figure 1c, the Qsearch algorithm finds a better linearly connected design with only six CNOT gates.

D. Blocks, Permutations, and Synthesis

Mapping and routing can be performed at the many-qubit gate level. The Orchestrated Trios [8] compiler shows that preserving Toffoli 3-qubit operations during qubit mapping and routing can reduce routing overhead. Wille et al. [44] introduce using output permutations for routing classical circuits composed of Toffoli gates. It's worth noting that classical reversible logic synthesis [34], [44] reduces the search space with garbage output bits. In practical quantum circuits, however, none of the output qubits are garbage. All these approaches leverage a fixed block implementation, e.g., unchangeable implementations of the Toffoli gate.

Applying the power of synthesis when reasoning about many-qubit blocks within a circuit provides the main insight of this paper: *Given a block, we need to consider arbitrary permutations of its input and output qubits to find the best quality implementation.*

Considering the 3-qubit QFT example, permutation-aware synthesis (PAS) further reduces the gate cost by finding the best output permutation. It implements this circuit with only five CNOTs, as shown in Figure 1d. To the best of our knowledge, this is the best-known implementation of this essential circuit. Note that the circuit maps (q_0, q_1, q_2) onto the permutation (q_1, q_0, q_2).

III. PAM Overview

An overview of the PAM framework is shown in Figure 2. First, PAM vertically partitions the input n-qubit quantum circuit into k-qubit blocks, B_1, ... , B_M, by grouping adjacent gates. Second, based on the previous discussion, we need to resynthesize each block for all possible mapping of input qubit permutations to output qubit permutations. This ensures we find the shortest possible implementation for some given topology. Additionally, We must resynthesize for all k-qubit topologies (sub-topologies) to ensure we have the best version for all potential placements on hardware. As shown in the previously mentioned example, 3-qubit blocks generally need four sub-topologies. One comes from the fully-connected or all-to-all 3-qubit architecture, and the other three represent all orientations of a 3-qubit line or nearest-neighbor architecture. The resynthesis results, including the associated qubit permutation, sub-topology, and circuit are stored for use during the next mapping phase.

The permutation-aware mapping algorithm continues over the partitioned circuit, unlike standard heuristical mappers, which mainly deal with native gates. To accomplish this, we supplement the SABRE algorithm with a novel heuristic to evaluate the current mapping state. Additionally, we add an extra processing step when moving gates from the unmapped to the mapped region. During this step, we utilize another novel heuristic to select the synthesized block permutation that best balances gate count and routing overhead for subsequent blocks. These block-level permutations leverage implicit communication buried in their computation to beneficially affect the state of the progressing mapping algorithm, drastically reducing the need for SWAP gates to perform global communication.

IV. Permutation-Aware Synthesis

We formalize here the concept of permutation-aware synthesis. Consider the example in Figure 3, where given U, a 3-qubit unitary, a synthesis algorithm constructs a circuit with three CNOTs, and the qubit ordering is preserved. However, we can consider alternate qubit permutations by inserting an input order permutation P_i and its inverse P_i^T. This insertion is allowed since P_i and P_i^T will cancel out, and the circuit's functionality will remain unchanged. Similarly, we can insert an output order permutation P_o and its inverse P_o^T.

After introducing these four extra permutations, we can group P_i, U, and P_o to generate a permuted unitary for synthesis. Since applying a gate on the left is equivalent to multiplying its unitary matrix on the right, the permuted unitary gate is represented as $P_o U P_i$. This permuted unitary operation may require fewer basic gates to implement. We can easily simulate the insertion of permutations ${P_i}^T$ and ${P_o}^T$ by classically reordering the qubit register and does not require any gates to implement.

As shown in Figure 3, the gate marked in yellow is a permuted unitary gate, which can be synthesized with only two CNOT gates. The permutations ${P_i}^T$ and ${P_o}^T$ have the effect of permuting the input and output qubit orders but can be handled classically by changing the index of the input and output qubit orders. In other words, permutations can be factored out to the inputs and outputs and resolved through classical processing. The core idea of our permutation-aware synthesis and permutation-aware mapping algorithm is the association of the original unitary with input and output permutations and the methodology to resolve the remaining permutations. We can then synthesize permutations to find the one that yields the fewest gates.

A. Permutation Search Space

For an n-qubit circuit, there are $n! \times n!$ input and output permutation combinations in total. We refer to the algorithm that evaluates all the input and output permutation combinations as FullPAS. To reduce the search space, we also introduce a sequential permutation-aware synthesis algorithm called SeqPAS. In SeqPAS, we first evaluate all the input permutations to find the best permutation P_i. Then, we fix the

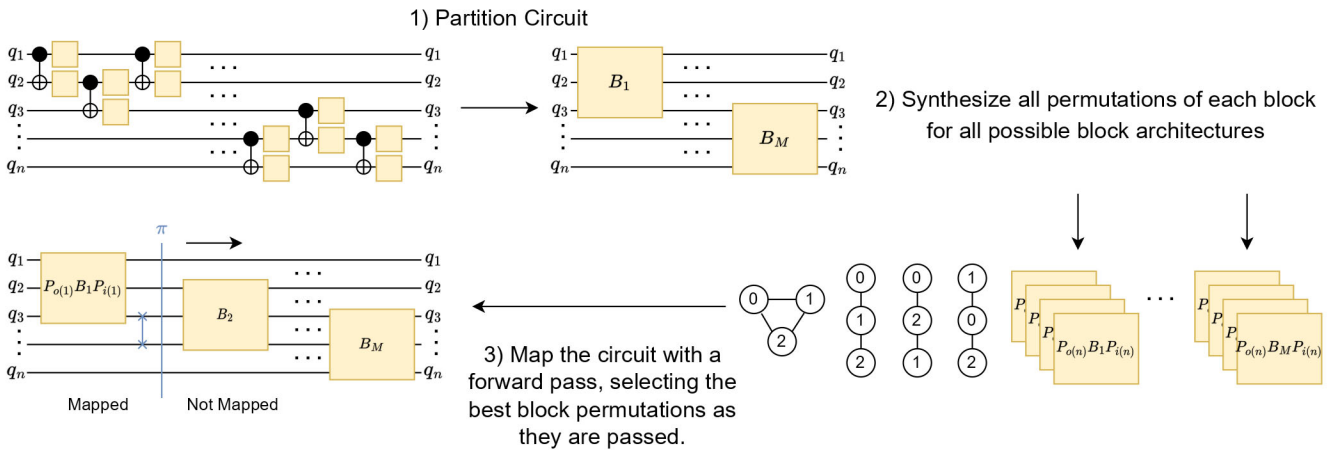


Fig. 2: *This example applies permutation-aware mapping on a circuit with 3-qubit blocks. Each block is synthesized with possible input and output permutations. Our permutation-aware mapping procedure then resolves the different qubit permutations as we map the blocks to the device. Additionally, as with any mapping or routing algorithm, inserting SWAP gates is necessary.*

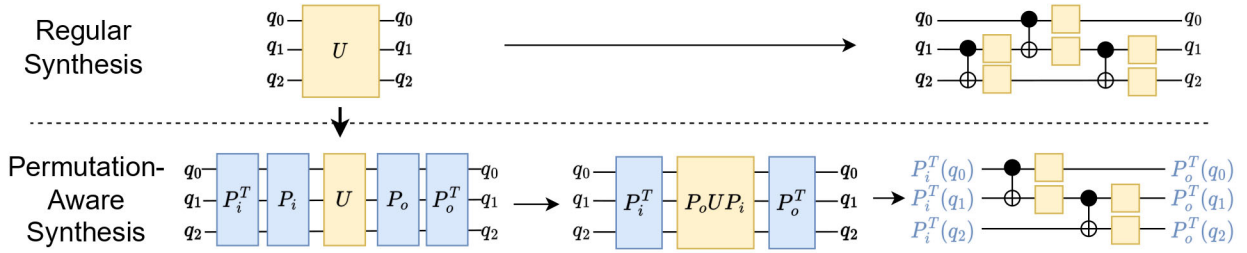


Fig. 3: *Regular synthesis will construct a circuit implementing a given unitary matrix preserving input and output qubit orderings. Before synthesizing a unitary, permutation-aware synthesis will factor out implicit qubit communication, leading to an overall shorter circuit. This action, however, will not preserve input and output qubit orderings and will require some simple classical processing when preparing the initial qubit state and reading out the final qubit state.*

best input permutation P_i to find the best output permutation P_o. The total number of evaluations in SeqPAS is $2 \times n!$. In Section VII-A, we will comprehensively compare these two PAS designs with the other synthesis and routing algorithms. In most cases, SeqPAS generates circuits with a gate count close to that of FullPAS and with much lower compilation overhead.

V. Permutation-Aware Mapping

In this section we clarify how we combine circuit partitioning, block-level permutation-aware resynthesis, and novel heuristics with tried-and-trued routing techniques to assemble our permutation-aware mapping framework. In addition to a target architecture, this process takes as input a logical circuit composed of native gates acting on any subset of qubits. It outputs a physical circuit consisting of the same native gates now only acting on valid sets of qubits allowed by the target.

Like other mapping algorithms, we break the problem into two steps: layout and routing. Layout discovers an initial logical to physical qubit mapping; routing then progresses this mapping through the circuit, updating it and adding SWAP gates as necessary to connect interacting logical qubits. While these two steps are distinct in our framework, the same circuit sweep methods that utilize circuit partitions and permutation-aware synthesis implement both. As such, we first describe our partitioning and resynthesis steps and then detail our heuristic circuit sweep. After proposing the full algorithm, we provide an analysis of PAM's computational complexity.

A. Circuit partitioning

The PAM algorithm first partitions a logical circuit vertically into k-qubit blocks. Vertical partitioning groups together gates acting on nearby qubits into blocks and is commonly implemented by placing gates into bins as a circuit is swept left to right. This method contrasts horizontal partitioning techniques [1] used in distributed quantum computing to best separate qubits. The binning approach to partitioning is excellent for our algorithm due to its scalability. These partitioners are linear with respect to gate count, $O(M)$, and we found that alternative partitioning techniques showed little variance in experimental results.

B. Permutation-aware resynthesis

After partitioning a circuit, we represent it as a sequence of k-qubit logical blocks containing the original gates. Later, layout and routing will replace these blocks with one of many permutated versions. Having all block permutations accessible enables our heuristic to compare the quality of each and select

the one that best balances its gate count with its effect on mapping. To discover all possible block permutations, we use permutation-aware synthesis.

If we perform permutation-aware synthesis online during mapping, we would serialize the required block synthesis. For very large circuits, this will become intractable very quickly. To overcome this, we perform the resynthesis step across all blocks in parallel offline. While this is an embarrassingly parallel problem, performing it offline has the added challenge of not knowing the block's final physical position and, therefore, its required topology. As a result, we will also need to synthesize for different topologies in addition to permutations.

We now resynthesize each block once for each possible permutation and connectivity requirement. We synthesize for all topologies because it allows us to identify extra connections provided by the hardware. For example, as shown in Figure 2, every three-qubit block will have six possible input permutations, six possible output permutations, and four different possible connectivities. Naively, this totals 144 synthesis calls for every three-qubit block.

We can dramatically reduce the number of required synthesis calls in two ways. First, we can perform a quick sub-topology check of the target architecture to eliminate possible connectivities. For example, if the target architecture is only linearly-connected, we do not need to consider the all-to-all connectivity requirement during block resynthesis. This is because no possible placement of a 3-qubit block on a linearly-connected topology can ever be fully-connected. Although not intuitive, it is common to eliminate some required sub-topologies when targetting realistic architectures with 3-qubit blocks.

The second way to reduce the number of synthesis calls is to recognize equivalent permutations. One can permute a resynthesized circuit to produce a new circuit implementing the same unitary with different input and output permutations and a rotated topology. Since there are $n!$ ways to permute a circuit, we can reduce the number of synthesis calls required for permutation-aware resynthesis by that many. After applying this optimization to 3-qubit blocks, we only need to synthesize a max of 24 different unitaries.

C. Heuristic circuit sweep

PAM's layout and routing algorithms utilize the same circuit sweep responsible for evolving a given logical-to-physical qubit mapping through a circuit. This section describes how we augment the SABRE algorithm [20] to leverage block-level permutations.

We follow the SABRE convention in dividing the logical circuit into a front layer F and E, an extended set. The front layer consists of gates with no predecessors, and the extended set consists of the first $|E|$ successors of the front layer, where $|E|$ is configurable. The extended layer E is defined for lookahead analysis. As the sweep builds the physical circuit, it removes gates from the logical circuit and updates F and E.

In our first change from the SABRE algorithm, we generalized the heuristic cost function from [52] to support arbitrary-sized gates given by:

$$\mathcal{F}(\pi) = \frac{1}{|F|} \sum_{b \in F} \sum_{i,j \in b} D[\pi(b.i)][\pi(b.j)]$$

$$\mathcal{E}(\pi) = \frac{W_E}{|E|} \sum_{b \in E} \sum_{i,j \in b} D[\pi(b.i)][\pi(b.j)]$$

$$H(\pi) = \mathcal{F}(\pi) + \mathcal{E}(\pi)$$

Here b is a gate block. D is the distance matrix that records the distance between physical qubits. $|F|$ and $|E|$ are the size of the front and extended layers, respectively. Minimizing this heuristic requires bringing all front layer gates' logical qubits physically closer together. To add lookahead capabilities, the operations in the extended set also contribute a term weighted by a configurable value W_E.

It is essential to note some challenges with heuristic mapping algorithms when generalizing from two-qubit to many-qubit gates. There are many ways to bring more than two qubits together on a physical architecture, creating many local minimums in a heuristic swap search. To combat this, we disabled swaps between any pair of logical qubits if an operation exists in the front layer containing both.

The second change we make to the SABRE algorithm is adding a step when removing an executable gate from the front layer and placing it in the physical circuit. In our case, the gates are blocks, and we have already pre-synthesized their permutations. The current mapping determines the block's input permutation and sub-topology, leaving the block's output permutation to be freely chosen. For 3-qubit blocks, we will have six possible choices for output permutation.

Two factors determine which output permutation to select for a given block. The chosen permutation will alter the ongoing mapping process potentially for the better. Also, the circuits associated with each permutation will have differing gate counts. We want to choose an output permutation that balances the resulting block's gate count with the overall effect on mapping. We modify the swap search heuristic to select the best permutation, producing the following heuristic:

$$P(\pi) = W_P \times C[b][G_b][(P_i, P_o)] + H(P_o(P_i(\pi)))$$

Here $C[b][G_b][(P_i, P_o)]$ is the 2-qubit gate count for the block b with subtopology G_b and permutations (P_i, P_o). The W_P weights the gate cost with the mapping cost and has been empirically discovered to be 0.1. Note that after applying the permuted block, the mapping cost function is evaluated using mapping updated by both input and output permutations.

In summary, our circuit sweep iterates over a partitioned circuit inserting swaps according to a swap search with a generalized heuristic to make blocks in the front layer executable. At this point, they are moved to the physical circuit and assigned a permutation according to a novel heuristic that updates the mapping state as the algorithm advances.

D. Layout and routing

Both of PAM's layout and routing algorithms are built trivially using the circuit sweep method previously described. Similar to SABRE, layout is conducted by randomly starting with an initial mapping and evolving it via the heuristic circuit sweep. Once complete, layout evolves the resulting mapping through the reverse of the logical circuit. This back-and-forth process is repeated several times until a stable mapping has been discovered. Routing then performs a single forward pass of the circuit sweep starting from the mapping that layout found.

Some corner cases exist where the heuristic may not select the best permutation. After routing the circuit, we can catch these corner cases by repartitioning and resynthesizing the circuit. The repartitioning process will group newly placed SWAP gates with other operations. This process is termed as gate absorption in some prior works [27], [40]. However, these works primarily discussed the absorption of SWAP gates with SU(4) gates. In our case, repartitioning and resynthesis of many-qubit blocks and swap networks allow us to reduce circuit gate count further.

E. Complexity analysis

The PAM framework is scalable in terms of both the number of qubits N and the total 2-qubit gate count M. It has the same level of time complexity as SABRE, which is $O(N^{2.5}M)$.

The PAM framework consists of four compilation steps. First, a circuit is partitioned into gate blocks with the partitioning algorithm. The default quick partition algorithm [47] in BQSKit has complexity of $O(M)$. Second, we use PAS to synthesize the permutations for each block. Since we limit the block size to less than three, the synthesis time for each block is bounded by a constant time limit $O(C)$. In the worst case, the total number of block equals the total number of gates M over the constant block size. Therefore, the PAS step has time complexity of $O(M)$. The layout step and the routing step in the worst case have the same time complexity as does the SABRE routing algorithm, $O(N^{2.5}M)$. By adding all the steps together, the PAM framework has time complexity of $O(N^{2.5}M)$, which is as scalable as that of other heuristic routing algorithms.

VI. Experimental Setup

The permutation-aware synthesis and mapping algorithms are implemented by using the BQSKit framework [47]. We compared the proposed algorithms with the original SABRE algorithm and three industrial compilers: Qiskit [7], TKET [37], and BQSKit. When possible, we additionally compared the algorithms with an optimal mapping algorithm OLSQ [39] followed by Qiskit optimizations.

A. Benchmarks

We used two sets of benchmarks to evaluate the proposed permutation-aware algorithms. When evaluating algorithms at the block level, we used a collection of small 3-, and 4-qubit circuits, which are either commonly used as building blocks in larger quantum programs or represent a smaller version of standard programs. These are listed in Figure 4a. Qiskit generated all of them except for the QAOA circuit, which was generated by Supermarq [41]. The Toffoli and Fredkin gates are well studied, and often compilers will be able to handle them through optimized workflows. To ensure a diverse benchmark set, we included some less-optimized gates: the singly and doubly controlled-MS XX gate [25]. QFT and QAOA circuits were included because they have been used extensively in past benchmark sets. Supermarq [41] generated the 4-qubit, fermionic-SWAP QAOA circuit.

Benchmark	CNOT Gates
ccx3	6
cswap3	8
cxx3	22
ccxx4	118
qft3	6
qft4	12
qaoa4	18

(a) small block benchmarks

Benchmark	CNOT Gates
adder63	1405
mul60	11405
qft5	20
qft64	1880
grover5	48
hub18	3541
shor26	21072
qaoa12	198
tfim64	4032
tfxy64	4032

(b) large quantum benchmarks

Fig. 4: *Two-qubit gate counts for the small block and large quantum program benchmark suites. The number of qubits in the circuit is given as a suffix.*

To evaluate the qubit mapping and circuit optimization capabilities of our proposed algorithm against full-scale compilers, we used a benchmark suite consisting of 10 real quantum programs of various types ranging in size from 5 to 64 qubits. We included two commonly used arithmetic circuits [5], [45], which contain long chains of 2-qubit gates. These chains are worst-case scenarios for partitioning compilers and are useful to evaluate. We included a 5-qubit Grover and 26-qubit Shor circuit generated by Qiskit [11], [36]. The suite also included two variational quantum algorithms: Supermarq's 12-qubit fermionic-SWAP QAOA circuit [9], [41] and an 18-qubit circuit simulating a spinful Hubbard model generated with OpenFermion [13], [23]. Moreover, we included two real-time evolution circuits: a transverse-field ising (TFIM) [35] and a transverse-field XY (TFXY) model. The constant-depth F3C++ compiler [2], [3], [18] produced these circuits, which before PAM were the best implementations. Figure 4b lists all large quantum program benchmarks alongside their gate counts.

B. Experiment platform

All experiments were executed with Python 3.10.7 on a 64-core AMD Epyc 7702p system with 1 TB of main memory running Ubuntu 20.04 as the operating system. We used versions 1.0.3, 0.38.0, 1.6.1, and 0.0.4.1 for the BQSKIT, Qiskit, PyTKET, and OLSQ packages, respectively.

C. Algorithm configuration

Unless otherwise specified, we used the Qsearch algorithm for 3-qubit synthesis and the LEAP algorithm for 4-qubit

	ccx	cswap	cxx	qft3	qft4	qaoa4	ccxx
Qiskit	9	10	31	7	17	18	270
TKET	9	10	29	9	21	17	172
OLSQ+Opt	9	10	21	9	17	18	184
Qsearch	8	8	5	6	16	18	15
SeqPAS	7	8	4	6	14	14	13
FullPAS	7	8	4	5	13	12	10

(a) CNOT counts

	ccx	cswap	cxx	qft3	qft4	qaoa4	ccxx
Qiskit	2.43	2.43	2.61	2.41	2.51	2.46	3.76
TKET	0.05	0.07	0.16	0.07	0.13	0.15	0.92
OLSQ+Opt	2.66	2.66	5.51	2.68	3.40	2.72	19325.55
Qsearch	10.21	7.51	1.78	3.23	90.45	216.25	54.33
SeqPAS	23.73	29.45	7.47	7.98	5188.93	2705.54	5974.15
FullPAS	23.83	83.38	7.49	9.80	46733.13	16582.57	36676.70

(b) Compile time in seconds

Fig. 5: *Common quantum circuit building blocks compiled to a linear topology using varying methods.*

	ccx	cswap	cxx	qft3	qft4	qaoa4	ccxx
Qiskit	6	7	17	6	12	18	114
TKET	6	7	17	6	12	12	95
OLSQ+Opt	6	7	17	6	12	18	114
Qsearch	6	7	5	6	12	13	11
SeqPAS	6	7	5	6	13	12	9
FullPAS	6	7	4	5	10	9	9

(a) CNOT counts

	ccx	cswap	cxx	qft3	qft4	qaoa4	ccxx
Qiskit	2.42	2.39	2.47	2.40	2.45	2.44	3.20
TKET	0.06	0.07	0.15	0.07	0.13	0.15	0.92
OLSQ+Opt	2.61	2.61	2.81	2.59	2.75	2.73	5.27
Qsearch	7.40	11.34	2.61	6.45	1683.54	2752.78	336.55
SeqPAS	19.73	46.65	5.88	9.79	4117.27	1793.36	1232.44
FullPAS	25.64	65.95	8.85	13.46	234174.05	101642.08	20775.23

(b) Compile time in seconds

Fig. 6: *Common quantum circuit building blocks compiled to a fully-connected topology using varying methods.*

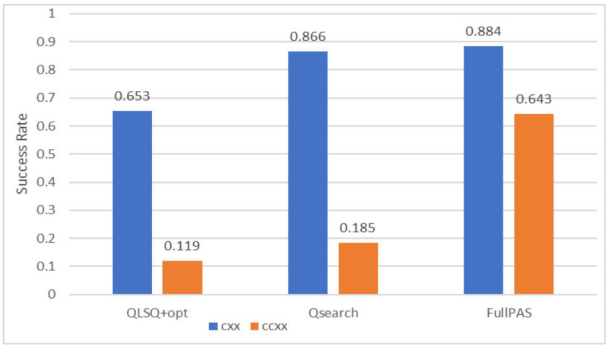


Fig. 7: *Comparison of OLSQ+Opt, Qsearch, and FullPAS on* `ibm_oslo`

synthesis. For both, we used the BQSKit implementation configured with the recommended settings: 4 multistarts and the default instantiater with a success threshold of 10^{-10}. The default BQSKit partitioner handled all circuit partitioning.

Similarly to the original SABRE evaluation, we configured PAM with a maximum extended set size $|E|$ of 20 and a weight W_E of 0.5. We used a decay delta of 0.001 and reset the decay every five steps or after mapping a gate. When discovering the initial layout, we performed two complete forward-and-backward passes. PAM's gate count heuristic weight W_P is set to 0.1. We used the BQSKit implementation and the same values for common parameters when evaluating the original SABRE algorithm. For the Qiskit, BQSKit, and TKET compilers we used the recommended settings with maximum optimization level.

The experimental results are verified with classical simulation and numerical instantiation based error upper-bound verification [30], [46]. The error upper bounds on all outputs were less than 10^{-8}.

VII. Evaluation

A. Block mapping

We first evaluated the mapping and optimization potential for synthesis and our permutation-aware synthesis framework at the block level. We selected two architectures to evaluate the different methods: a line with only nearest-neighbor connectivity and a fully connected topology. Figures 5 and 6 respectively detail the final CNOT counts and total compile time for the two different target architectures.

Fully permutation-aware synthesis (FullPAS) produced shortest circuits in all cases. FullPAS built circuits with an average of 42%, 43%, 42%, and 21% fewer gates than did Qiskit, TKET, OLSQ, and QSearch, respectively, where SeqPAS produced circuits with an average of 37%, 37%, 36%, and 12% fewer gates.

An optimal decomposition is not always precomputed and available or trivial to compute by hand, however, as in the case of the controlled MS gates. FullPAS resulted in a cxx circuit with 19% and 24% of the gates in the best nonsynthesized result when compiling to the linear or fully connected topology, respectively. This improvement is even more pronounced in the case of the ccxx circuit, where FullPAS produced circuits with as much as 27 times fewer gates; however, improvements over Qsearch are much more modest. Nonetheless, these modest gains are still significant. FullPAS compiled a 5-CNOT qft3 circuit for all topologies; this is, to the best of our knowledge, the new best-known implementation of this essential circuit.

These significant improvements in quality require many synthesis calls and, as a result, more runtime than other methods require. Since FullPAS calls for synthesizing all pairs of input and output permutations, its scaling is limited. SeqPAS, however, is much more palatable, with an average runtime of 24.25 seconds for 3-qubit blocks and 3175 seconds for 4-qubit blocks.

We evaluate the cxx and ccxx benchmarks on a 27-qubit `ibm_oslo` computer. The gate counts are reported in Fig-

TABLE I: Mapping and optimizing a quantum circuit benchmark suite to a fully connected topology.

	SABRE		Qiskit		TKET		BQSKit		PAM3	
benchmark	$\#CX$	time(s)	$\#CX$	time(s)	$\#CX$	time(s)	$\#CX$	time(s)	$\#CX$	time(s)
adder63	1405	3.23	1405	9.98	484	14.45	1195	34.41	**442**	187.08
mul60	11405	24.09	11403	72.27	4144	428.55	9926	225.75	**3938**	1493.63
qft5	20	0.28	20	2.42	20	0.49	20	4.04	**18**	18.59
qft64	1880	3.78	1720	10.74	1784	24.61	1771	188.87	**1665**	771.31
grover5	48	0.35	48	2.69	46	0.79	48	10.82	**44**	51.80
hub18	3541	6.87	3529	22.86	**3428**	76.35	3498	50.59	3459	524.00
shor26	21072	42.01	21072	109.30	20884	836.27	16319	1020.94	**14950**	9976.45
qaoa12	198	0.58	198	3.15	132	2.03	191	8.43	**129**	75.93
tfim64	4032	9.79	4030	31.17	4032	107.38	4013	169.91	**2820**	2232.45
tfxy64	4032	9.84	4032	31.00	4032	108.84	4014	170.04	**3294**	1791.33

TABLE II: Mapping and optimizing a quantum circuit benchmark suite targeting Rigetti's Aspen M2 chip

	SABRE		Qiskit		TKET		BQSKit		PAM3	
benchmark	$\#CX$	time(s)	$\#CX$	time(s)	$\#CX$	time(s)	$\#CX$	time(s)	$\#CX$	time(s)
adder63	3931	6.62	3250	23.90	1798	15.51	3801	85.62	**1566**	301.63
mul60	30386	38.24	24832	196.47	14708	441.12	25580	514.29	**11172**	2400.01
qft5	41	0.39	34	2.68	35	0.26	29	4.18	**28**	24.23
qft64	6383	10.38	5107	34.19	4970	25.40	5575	293.78	**3861**	1194.87
grover5	108	0.49	110	3.05	82	0.57	63	13.57	**59**	89.74
hub18	15151	10.58	13031	67.51	**11680**	77.20	12236	187.58	11785	1089.22
shor26	44907	33.39	39171	220.17	46192	862.52	32110	795.92	**29055**	15528.36
qaoa12	303	0.55	**198**	4.34	253	1.96	219	13.29	302 (**188**)	100.12
tfim64	8403	25.13	**4032**	95.63	**4032**	109.26	6040	170.96	4532 (**2804**)	4014.57
tfxy64	8403	24.52	**4032**	138.95	**4032**	110.24	7884	292.50	5963 (**3319**)	5577.72

The numbers in brackets represent the experimental results of PAM3 with extra isomorphism check.

ure 5. As shown in Figure 7, FullPAS generates the circuit that has the highest success rate. In the ccxx example, permutation-aware synthesis reduces the CNOT gate count from 15 to 10, resulting in a 3.5x success rate boost.

B. Large circuits

To evaluate the mapping methods, we chose four real quantum architectures implemented in state-of-the-art quantum processors: Rigetti's Aspen M2 80-qubit chip [33], Google's 72-qubit Bristlecone chip [10], IBM 127-qubit Eagle chip [14], and a 64-qubit fully-connected topology similar to trapped-ion architectures [15], [31].

The 3-qubit version of the PAM algorithm (PAM3) produced the shortest circuits in the most trials, with an average of 35%, 18%, 9%, and 21% fewer gates than SABRE, Qiskit, TKET, and BQSKit. The results are demonstrated in Table I,II,III,IV. The optimal OLSQ mapper cannot find any solution for the benchmarks with tens of qubits, therefore we exclude it from the large circuit comparison. PAM3 built the shortest circuit in 29 out of the 40 trials (10 circuits and 4 architectures) or 37 out of 40 with an isomorphism check added.

QAOA, TFIM, and TFXY: In eight of the eleven times PAM3 produced a worse circuit, the benchmark was either a QAOA, TFIM, or TFXY circuit. This result is due to placement. These three circuits all require only linear connections, and theoretically, they can be mapped to all four chips without routing. Qiskit and TKET do a subgraph isomorphism check, which sometimes catches a perfect placement. This extra check highlights the downside of comparing our experimental mapping algorithm to complete commercial compilers. However, in the cases where they did not catch the isomorphism, PAM produced shorter circuits. Additionally, integrating the same isomorphism check can outperform them because we can often further reduce the circuit depth on a line. For example, suppose we pick a perfect placement and map the QAOA to a line with PAM3. In that case, we get a result with 188 CNOTs, which can be directly placed on any of the four experiment architectures and is shorter than all other compilers' output. Similarly, TFIM and TFXY can be compiled with 2804 and 3319 CNOTs by adding the isomorphism check. The isomorphism check only takes tens of seconds which is negligible. PAM3 produced the shortest circuit in 37 of the 40 trials by adding an isomorphism check. In the tables, The isomorphism check data is presented in parentheses.

C. Comparison with optimal layout solver

In this section, we compare PAM's solution quality and compilation time with the optimal solver OLSQ to evidence the effectiveness of permutation-aware mapping. Table V demonstrates the final gate count and the compile time. We use OLSQ for routing, followed by Qiskit optimizations. OLSQ finds the optimal mapping and routing that minimizes the number of inserted SWAP gates; however, since PAM directly synthesizes the unitary based on hardware connectivity, the resulting circuit is, on average, 10.7% smaller than OLSQ. Moreover, the optimal solver has scalability issues. It cannot find any solution on the coupling map of Google's Bristlecone. As shown in the table, when compiled with limited backend connectivity(Aspen-M2, IBM-Eagle), PAM has a shorter compilation time than OLSQ for most benchmarks.

D. Scaling beyond the NISQ era

To evaluate the scalability of the mapping algorithms past the capabilities of quantum hardware today, we generated a set

TABLE III: Mapping and optimizing a quantum circuit benchmark suite targeting Google's Bristlecone chip

	SABRE		Qiskit		TKET		BQSKit		PAM3	
benchmark	$\#CX$	time(s)	$\#CX$	time(s)	$\#CX$	time(s)	$\#CX$	time(s)	$\#CX$	time(s)
adder63	3274	6.79	2726	21.22	1326	15.89	2755	65.79	**925**	297.99
mul60	24974	32.35	20014	171.35	11989	437.34	18396	361.08	**9169**	2404.43
qft5	35	0.30	30	2.54	32	0.26	31	3.49	**22**	25.65
qft64	5153	9.14	4304	30.27	4175	25.30	4262	228.93	**3624**	1195.86
grover5	108	0.36	96	3.03	82	0.58	85	3.85	**62**	81.08
hub18	11227	8.89	10137	55.49	9084	77.05	9064	124.58	**8682**	1095.16
shor26	38241	29.86	36365	204.72	38070	849.95	28624	659.49	**24021**	15547.82
qaoa12	198	0.36	198	64.47	237	1.94	**205**	12.03	243 (**188**)	95.88
tfim64	6591	24.35	4828	80.56	4773	156.35	5187	173.95	**4344** (**2804**)	4312.13
tfxy64	6591	24.13	5204	78.47	**4773**	158.04	5814	255.63	4778 (**3319**)	5825.42

TABLE IV: Mapping and optimizing a quantum circuit benchmark suite targeting IBM's Eagle chip

	SABRE		Qiskit		TKET		BQSKit		PAM3	
benchmark	$\#CX$	time(s)	$\#CX$	time(s)	$\#CX$	time(s)	$\#CX$	time(s)	$\#CX$	time(s)
adder63	4906	9.08	4172	34.09	2318	16.02	4070	107.74	**1827**	316.22
mul60	37982	44.92	31284	349.58	18000	442.01	30817	612.56	**14553**	2493.17
qft5	41	1.37	35	2.72	38	0.26	32	6.26	**28**	60.02
qft64	6491	11.31	5760	46.52	5682	26.00	5511	321.04	**4466**	1190.36
grover5	114	1.38	122	3.14	82	0.58	60	12.00	**59**	79.73
hub18	17692	12.84	16990	93.66	**13648**	77.91	14288	222.41	14365	1161.29
shor26	50334	40.52	43705	239.15	54156	858.67	35659	978.47	**34205**	15684.63
qaoa12	309	1.54	**198**	3.38	241	1.81	276	12.60	232 (**188**)	96.14
tfim64	12126	41.26	**4032**	402.50	**4032**	107.16	8730	241.97	10652 (**2804**)	4493.52
tfxy64	12126	40.90	**4032**	395.86	**4032**	108.33	9469	297.04	8260 (**3319**)	5852.39

TABLE V: Quality of solutions and compile time (s) of OLSQ + opt and PAM3

	Fully-connected				Aspen M2				IBM Eagle			
	OLSQ		PAM3		OLSQ		PAM3		OLSQ		PAM3	
benchmark	$\#CX$	time(s)	$\#CX$	time(s)	$\#CX$	time(s)	$\#CX$	time(s)	$\#CX$	time(s)	$\#CX$	time(s)
alu-v0	17	2.8	**13**	15.17	28	207.36	**21**	27.89	28	324.10	**21**	27.99
qft5	20	1.58	**18**	18.59	**28**	12.20	**28**	24.23	**28**	11.86	**28**	60.02
grover5	48	2.25	**44**	51.80	76	393.67	**59**	89.74	76	352.52	**59**	79.73
qaoa8	24	1.98	**23**	8.1	38	66.18	**35**	9.33	**45**	666.38	47	11.75

of QFT circuits ranging from 128 to 1024 qubits and mapped them to a proposed heavy-hexagonal chiplet architecture [19]. We built an architecture following the tree-of-grids approach with a 3-node tree containing a 4×4-grid of 27-qubit chiplets. The results are shown in Figure 8. PAM always generates the circuit with the fewest gate count; for the 1024 qubit QFT algorithm, PAM generates the shortest circuit with 206310 CNOTs, with 8159 CNOT gate reduction compared to the next best result from TKET. As the number of qubits increases, the gaps between the compilation time of PAM and other compilers are narrowing. This highlights the scalability of our routing framework and the capability to handle future hardware designs. Furthermore, PAM's synthesis step is embarrassingly parallel and can be sped up with more computing power. To demonstrate this, we reran the benchmarks with 16 Nodes of NERSC's Perlmutter supercomputer and plotted the new times alongside the others.

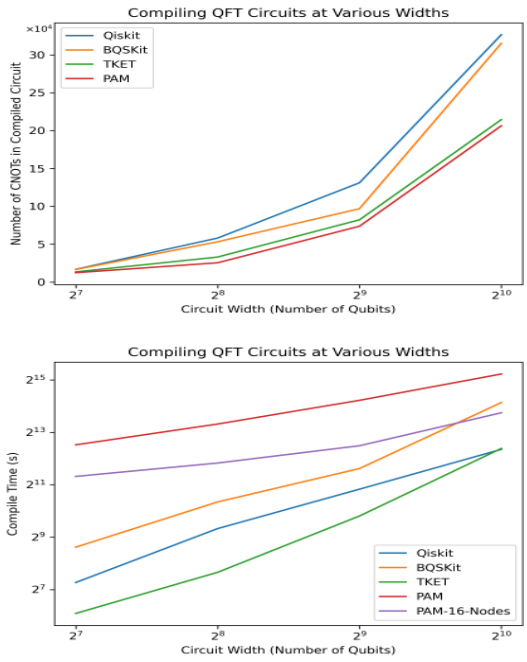


Fig. 8: *Scaling of the QFT benchmark*

E. Closer examination of the improvements

Since we have introduced a few features that improve upon the original SABRE algorithm, we thought it necessary to analyze how much each improves individually. In Figure 9 each additional feature is measured separately when compiling the multiply circuit. The PrePAM and PostPAM represent the cases where we only enable permutation on the input or output sides. We start with the original SABRE algorithm and then

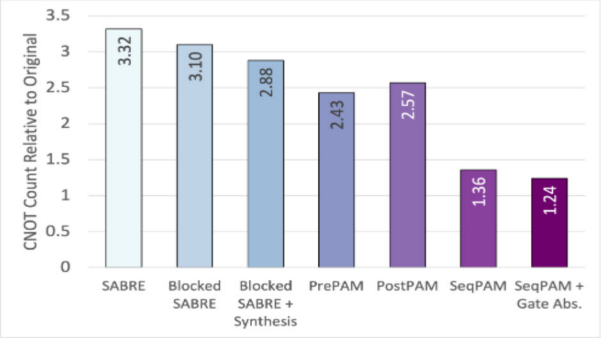


Fig. 9: *A breakdown of the improvements each individual feature adds on top of the SABRE algorithm. These results are from compiling the 60-qubit multiply circuit to the IBM Eagle architecture.*

introduce the concept of partitioning. Just by mapping blocks in a circuit rather than gates, we can see an improvement which we believe is because this increases the lookahead factor of the SABRE. Using synthesis to route inside the blocks improves the results. When we introduce the concept of permutation-aware-synthesis, we see the next big jump even if it is just one-sided with PrePAM and PostPAM. Furthermore, doing both sides in SeqPAM introduces the biggest jump. Finally, gate absorption further improves the result.

VIII. Discussion

A. Relevance to trapped ions

We have mentioned that our permutation-aware algorithms can leverage hardware connectivity by design. This effect is visible when compiling the linearly connected tfim64 and tfxy64 circuits to the fully connected topology. No other compiler can effectively utilize the full-connectivity by design; however, PAM3 produces a circuit with 2,820 CNOTs versus the 4,032 tfim64 input. The next best is BQSKit with 4,013 CNOTs. These TFIM input circuits were previously the best-known implementations of these real-time evolution circuits.

One way to quantify this concept is by using Supermarq's [41] program communication metric. The metric measures how sparsely or densely a circuit's logical connectivity is. A program communication value of 0 implies no connectivity, while a value of 1 implies that every qubit requires a connection with every other qubit. The 12-qubit QAOA started with a communication score of 0.167 but ended with a score of 1. This shift implies that we took the linearly connected input and returned a fully connected output with fewer CNOTs than any other compiler. Additionally, the scores improved in all the other cases when compiling to an all-to-all architecture and in most cases with the densely connected Bristlecone architecture. Increasing program communincation has particular significance for trapped-ion architectures. This class of quantum processors allows a program to apply a gate to any two pairs of qubits. PAM's ability to fully leverage the hardware connectivity is advantageous as an optimization pass for these architectures.

B. Building PAM into a workflow

PAM3 produced circuits shorter than state-of-the-art compilers in many trials tested; however, PAM3 is just a mapping algorithm with good optimization potential. We can replace the mapping algorithm inside Qiskit, TKET, and BQSKit and sum up to a better compiler. We did this and compiled the qft64 to the M2 chip and saw an additional reduction of 15%, 55%, and 13% CNOTs when compiling with Qiskit, TKET, and BQSKit, respectively.

C. Tunability

PAM3 has built efficient circuits, but it always tends to take a lot more time than other compilers. Algorithm scientists will spend the time necessary to produce the best circuit possible, mainly since circuits are often compiled only once, quantum computer time is expensive, and longer circuits are more likely to produce erroneous results. Our proposed algorithm has many parameters one can adjust to improve runtime. In particular, the number of multistarts for instantiation has the most significant impact on runtime. For example, if we decrease the number of multistart to one, the runtime of the the shor26 reduces from 15547.82 seconds to 5908.40 seconds for the Bristlecone architecture.

IX. Conclusion

In this work we built on top of both general unitary synthesis and heuristic-based mapping algorithms by introducing the idea of permutation awareness with respect to the mapping problem. This codesign was accomplished by first lifting mapping from the native gate level to the block level. This elevation led to generally good results on its own but also introduced many new opportunities for optimization. While we have shown that these algorithms are effective and competitive, we have demonstrated the ability to leverage hardware connectivity is particularly helpful for optimizing the circuits for fully connected architectures. We have also shown the implementability and tunability of our algorithms with potential application in existing compiler frameworks.

Acknowledgements

This work was supported by the DOE under contract DE-5AC02-05CH11231 and DE-AC02-06CH11357, through the Office of Advanced Scientific Computing Research (ASCR), under the Accelerated Research in Quantum Computing (ARQC) program. This research also used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231 using NERSC award DDR-ERCAPm4141.

References

[1] J. M. Baker, C. Duckering, A. Hoover, and F. T. Chong, "Time-sliced quantum circuit partitioning for modular architectures," in *Proceedings of the 17th ACM International Conference on Computing Frontiers*, 2020, pp. 98–107.

[2] L. Bassman, R. Van Beeumen, E. Younis, E. Smith, C. Iancu, and W. A. de Jong, "Constant-depth circuits for dynamic simulations of materials on quantum computers," *Materials Theory*, vol. 6, no. 1, pp. 1–18, 2022.

[3] D. Camps, E. Kökcü, L. Bassman Oftelie, W. A. De Jong, A. F. Kemper, and R. Van Beeumen, "An algebraic quantum circuit compression algorithm for hamiltonian simulation," *SIAM Journal on Matrix Analysis and Applications*, vol. 43, no. 3, pp. 1084–1108, 2022.

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

[5] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton, "A new quantum ripple-carry addition circuit," *arXiv preprint quant-ph/0410184*, 2004.

[6] M. G. Davis, E. Smith, A. Tudor, K. Sen, I. Siddiqi, and C. Iancu, "Towards optimal topology aware quantum circuit synthesis," in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2020, pp. 223–234.

[7] Q. Developers, "Qiskit: An Open-source Framework for Quantum Computing," Jan. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.2562111

[8] C. Duckering, J. M. Baker, A. Litteken, and F. T. Chong, "Orchestrated trios: compiling for efficient communication in quantum programs with 3-qubit gates," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 375–385.

[9] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.

[10] "A Preview of Bristlecone, Google's New Quantum Processor," https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html, accessed: 2020-10-09.

[11] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.

[12] T. Häner, T. Hoefler, and M. Troyer, "Assertion-based optimization of quantum programs," *arXiv preprint arXiv:1810.00375*, 2018.

[13] J. Hubbard, "Electron correlations in narrow energy bands," *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 276, no. 1365, pp. 238–257, 1963.

[14] "Ibm quantum breaks the 100-qubit processor barrier," https://research.ibm.com/blog/127-qubit-quantum-processor-eagle, accessed: 2022-11-20.

[15] "Unveiling ionq forte: The first software-configurable quantum computer," https://ionq.com/posts/may-17-2022-ionq-forte, accessed: 2022-05-30.

[16] R. Iten, R. Moyard, T. Metger, D. Sutter, and S. Woerner, "Exact and practical pattern matching for quantum circuit optimization," *ACM Transactions on Quantum Computing*, vol. 3, no. 1, pp. 1–41, 2022.

[17] T. Itoko, R. Raymond, T. Imamichi, and A. Matsuo, "Optimization of quantum circuit mapping using gate transformation and commutation," *Integration*, vol. 70, pp. 43–50, 2020.

[18] E. Kökcü, D. Camps, L. B. Oftelie, J. K. Freericks, W. A. de Jong, R. Van Beeumen, and A. F. Kemper, "Algebraic compression of quantum circuits for hamiltonian evolution," *Physical Review A*, vol. 105, no. 3, p. 032420, 2022.

[19] N. LaRacuente, K. N. Smith, P. Imany, K. L. Silverman, and F. T. Chong, "Short-range microwave networks to scale superconducting quantum computation," *arXiv preprint arXiv:2201.08825*, 2022.

[20] 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.

[21] 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.

[22] 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.

[23] J. R. McClean, N. C. Rubin, K. J. Sung, I. D. Kivlichan, X. Bonet-Monroig, Y. Cao, C. Dai, E. S. Fried, C. Gidney, B. Gimby *et al.*, "Openfermion: the electronic structure package for quantum computers," *Quantum Science and Technology*, vol. 5, no. 3, p. 034014, 2020.

[24] 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.

[25] K. Mølmer and A. Sørensen, "Multiparticle entanglement of hot trapped ions," *Phys. Rev. Lett.*, vol. 82, pp. 1835–1838, Mar 1999. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.82.1835

[26] L. d. Moura and N. Bjørner, "Z3: An efficient SMT solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[27] 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.

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

[29] 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.

[30] T. Patel, E. Younis, C. Iancu, W. de Jong, and D. Tiwari, "QUEST: systematically approximating quantum circuits for higher output fidelity," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 514–528.

[31] "Quantinuum completes hardware upgrade; achieves 20 fully connected qubits," https://www.quantinuum.com/news/quantinuum-completes-hardware-upgrade-achieves-20-fully-connected-qubits, accessed: 2022-06-30.

[32] P. Rakyta and Z. Zimborás, "Efficient quantum gate decomposition via adaptive circuit compression," 2022. [Online]. Available: https://arxiv.org/abs/2203.04426

[33] "Rigetti systems aspen-m-2 quantum processor," https://qcs.rigetti.com/qpus, accessed: 2022-10-20.

[34] M. Saeedi, M. Sedighi, and M. S. Zamani, "A library-based synthesis methodology for reversible logic," *Microelectronics Journal*, vol. 41, no. 4, pp. 185–194, 2010.

[35] D. Shin, H. Hübener, U. De Giovannini, H. Jin, A. Rubio, and N. Park, "Phonon-driven spin-floquet magneto-valleytronics in mos2," *Nature communications*, vol. 9, no. 1, pp. 1–8, 2018.

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

[37] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "t— ket¿: a retargetable compiler for NISQ devices," *Quantum Science and Technology*, vol. 6, no. 1, p. 014003, 2020.

[38] E. Smith, M. G. Davis, J. M. Larson, E. Younis, L. B. Oftelie, W. Lavrijsen, and C. Iancu, "Leap: Scaling numerical optimization based synthesis using an incremental approach," *ACM Transactions on Quantum Computing*, 2021.

[39] B. Tan and J. Cong, "Optimal layout synthesis for quantum computing," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[40] B. Tan and J. Cong, "Optimal qubit mapping with simultaneous gate absorption," *arXiv preprint arXiv:2109.06445*, 2021.

[41] T. Tomesh, P. Gokhale, V. Omole, G. S. Ravi, K. N. Smith, J. Viszlai, X.-C. Wu, N. Hardavellas, M. R. Martonosi, and F. T. Chong, "SupermarQ: A scalable quantum benchmark suite," *arXiv preprint arXiv:2202.11045*, 2022.

[42] R. R. Tucci, "An introduction to Cartan's KAK decomposition for QC programmers," *arXiv preprint quant-ph/0507171*, 2005.

[43] Y. S. Weinstein, M. Pravia, E. Fortunato, S. Lloyd, and D. G. Cory, "Implementation of the quantum Fourier transform," *Physical Review Letters*, vol. 86, no. 9, p. 1889, 2001.

[44] R. Wille, D. Große, G. W. Dueck, and R. Drechsler, "Reversible logic synthesis with output permutation," in *2009 22nd International Conference on VLSI Design*. IEEE, 2009, pp. 189–194.

[45] X.-C. Wu, M. G. Davis, F. T. Chong, and C. Iancu, "Reoptimization of quantum circuits via hierarchical synthesis," in *2021 International Conference on Rebooting Computing (ICRC)*, 2021, pp. 35–46.

[46] E. Younis and C. Iancu, "Quantum circuit optimization and transpilation via parameterized circuit instantiation," *arXiv preprint arXiv:2206.07885*, 2022.

[47] E. Younis, C. C. Iancu, W. Lavrijsen, M. Davis, E. Smith *et al.*, "Berkeley quantum synthesis toolkit (bqskit) v1," Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), Tech. Rep., 2021.

[48] E. Younis, K. Sen, K. Yelick, and C. Iancu, "Qfast: Conflating search and numerical optimization for scalable quantum circuit synthesis," in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2021, pp. 232–243.

[49] C. Zhang, A. B. Hayes, L. Qiu, Y. Jin, Y. Chen, and E. Z. Zhang, "Time-optimal qubit mapping," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 360–374.

[50] X. Zhou, S. Li, and Y. Feng, "Quantum circuit transformation based on simulated annealing and heuristic search," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4683–4694, 2020.

[51] P. Zhu, Z. Guan, and X. Cheng, "A dynamic look-ahead heuristic for the qubit mapping problem of NISQ computers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4721–4735, 2020.

[52] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the ibm qx architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1226–1236, 2018.