Using ZDDs in the mapping of quantum circuits

Kaitlin Smith   Mitchell Thornton
Quantum Informatics Research Group, SMU
Dallas, Texas, USA

Mathias Soeken   Bruno Schmitt   Giovanni De Micheli
Integrated Systems Laboratory, EPFL
Lausanne, Switzerland

A critical step in quantum compilation is the transformation of a technology-independent quantum circuit into a technology-dependent form for a targeted device. In addition to mapping quantum gates into the supported gate set, it is necessary to map pseudo qubits in the technology-independent circuit into physical qubits of the technology-dependent circuit such that coupling constraints among qubits acting in multiple-qubit gates are satisfied. It is usually not possible to find such a mapping without adding SWAP gates into the circuit. To cope with the technical limitations of NISQ-era quantum devices, it is advantageous to find a mapping that requires as few additional gates as possible. The large search space of possible mappings makes this task a difficult combinatorial optimization problem. In this work, we demonstrate how zero-suppressed decision diagrams (ZDDs) can be used for typical implementation tasks in quantum mapping algorithms. We show how to maximally partition a quantum circuit into blocks of adjacent gates, and if adjacent gates within a circuit do not share common mapping permutations, we attempt to combine them using parallelized SWAP operations represented in a ZDD. Boundaries for the partitions are formed where adjacent gates are unable to be combined. Within each partition block, ZDDs represent all possible mappings of pseudo qubits to physical qubits.

1 Introduction

Today's NISQ-era quantum devices [9] support some given set of single- and two-qubit quantum gates. While single-qubit operations can be executed on any of the physical qubits, two-qubit quantum gates can only be performed by a pair of qubits that share a physical connection. The set of permissible qubit pairs are referred to as the coupling constraints. One task in quantum compilation algorithms is the mapping of a quantum circuit or algorithm, a sequence of quantum operations, onto the physical qubits of the device such that all two-qubit operations are executed with respect to device coupling constraints. This task is not always possible without including additional gates in the circuit.

Finding an optimum solution that minimizes the number of additional gates is "NP-hard" [2]. In order to efficiently find a solution, several heuristics have been proposed [11, 14, 5, 7]. Past work in mapping algorithms for quantum circuits are reported in [3, 12] A common bottleneck in these heuristic methods is due to the large combinational search space resulting in numerous possible ways of mapping the gates to the device.

In this paper, we discuss how zero-suppressed decision diagrams (ZDDs, [8, 6]) can be used in mapping algorithms to combat combinational complexity. These data structures were selected because nearest neighbor couplings within quantum devices make the possible connections between qubits in a quantum circuit a sparse set, and ZDDs, as compared to other types of decision diagrams, are efficient at representing sparse sets. We show how to implement two specific problems which appear in several heuristics: (1) finding a maximal subcircuit partition that can be mapped without adding gates, and (2)

Submitted to:
QPL 2019

© Smith, Soeken, Schmitt, De Micheli, and Thornton
This work is licensed under the
Creative Commons Attribution License.

how to determine and choose among all possible SWAP circuits that can execute in parallel in order to extend a partition. Once a maximal subcircuit partition is determined, a pseudo to physical qubit permutation from that partition is used to map the quantum circuit to a technology platform.

Finding maximal partitions for quantum circuit mapping is solved using SAT in [5]. In contrast to the SAT-based solution that finds one possible mapping, the ZDD-based algorithm generates all possible mappings for a maximal partition. All solutions are represented implicitly by means of a decision diagram, which may be used to count all solutions, query some solutions, or compute the solution that minimizes some linear cost function. All such tasks can be performed in time that is linear with respect to the size of the ZDD. All possible parallel SWAP operations used to extend the size of a partition are also stored in a ZDD.

The algorithms reported in this paper are intended to serve as motivating examples to illustrate how ZDDs may be used as a data structure for implementing a mapping algorithm. These algorithms have been prototyped and evaluated on a set of benchmarks. Effectiveness of the ZDD mapping algorithms is determined by using them in a preprocessing step for the publically-available compilers developed for the IBM and Rigetti quantum devices.

2 Preliminaries

2.1 Graphs

An undirected graph $G = (V,E)$ consists of a set of vertices V and a set of edges $E \subseteq \binom{V}{2} = \{V' \subseteq V \mid |V'| = 2\}$. Given two undirected graphs $G_1 = (V_1,E_1)$ and $G_2 = (V_2,E_2)$, we say that G_1 is a subgraph of G_2, if there exists an injective function $\varphi : V_1 \to V_2$ such that $\{v,w\} \in E_1$ implies $\{f(v), f(w)\} \in E_2$.

Given an ordered sequence $S = s_1,\dots,s_n$, we define an ordered partition $S_1,\dots,S_l$ as a set of nonempty subsequences $S_i = s_{b_i}, s_{b_i+1},\dots,s_{e_i}$ such that $s_{b_1} = s_1$, $s_{e_l} = s_n$, and $s_{b_i} = s_{e_{i-1}+1}$ for all $1 < i \leq l$.

Example 1. *Let* $S = 3,5,1,2,8,2,3,4$. *Then* $S_1 = 3,5,1$, $S_2 = 2,8$, *and* $S_3 = 2,3,4$ *is an ordered partition of S.*

2.2 Zero-suppressed decision diagrams

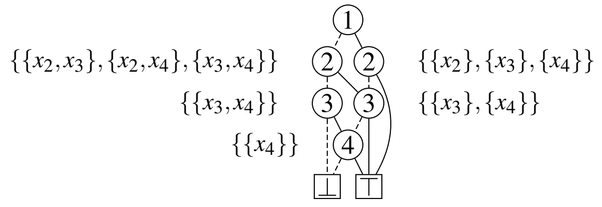
Given a set of variables $X = \{x_1,\dots,x_n\}$, a ZDD [8, 6] is a directed acyclic graph with nonterminal vertices N and two terminal vertices $\top$ and $\bot$. Each non-terminal vertex $v \in N$ is associated with a variable $V(v) \in \{1,\dots,n\}$ and two successor nodes $\textsc{hi}(v), \textsc{lo}(v) \in N \cup \{\top,\bot\}$. The nodes on a path to a terminal node follow a variable order. We have $\textsc{hi}(v) \in \{\top,\bot\}$ or $V(\textsc{hi}(v)) > V(v)$ for all v.[1] The same applies to $\textsc{lo}(v)$.

Each vertex in the ZDD represents a finite family of finite subsets over X. The terminal node $\bot$ represents the *empty family* $\emptyset$ and the terminal node $\top$ represents the *unit family* which is the set containing the empty set $\{\emptyset\}$. Each non-terminal v represents the subset

$$\textsc{lo}(v) \cup \{S \cup \{x_{V(v)}\} \mid S \in \textsc{hi}(v)\}. \tag{1}$$

A ZDD is *reduced* if there are no two vertices that represent the same sets. This implies that in a reduced ZDD there cannot be a vertex v with $\textsc{hi}(v) = \bot$, since such a vertex represents the set $\textsc{lo}(v)$. For the sake of convenience, we use ε_x to denote the *elementary family* $\{\{x\}\}$ for each $x \in X$. Finally, we use $\wp$ to refer to the ZDD that represents the *universal family* of all subsets of X.

[1] To simplify the presentation in the paper, we assume the variable ordering $1 < 2 < \cdots < n$. In practice, any permutation of this order can be used.



$\{\{x_2,x_3\},\{x_2,x_4\},\{x_3,x_4\}\}$
$\{\{x_3,x_4\}\}$
$\{\{x_4\}\}$
$\{\{x_2\},\{x_3\},\{x_4\}\}$
$\{\{x_3\},\{x_4\}\}$
1
2
2
3
3
4
$\perp$
$\top$

Figure 1: A ZDD representing the family of sets $\{\{x_1,x_2\},\{x_1,x_3\},\{x_1,x_4\},\{x_2,x_3\},\{x_2,x_4\},\{x_3,x_4\}\}$. All internal non-terminal nodes are annotated with the sets they represent.

We write $|f|$ to denote the number of sets in a family f. We write $Z(f)$ to denote the number of nodes, including the terminal nodes, of the reduced ZDD for f.

Example 2. *Fig. 1 shows a ZDD for* $f = \{\{x_1,x_2\},\{x_1,x_3\},\{x_1,x_4\},\{x_2,x_3\},\{x_2,x_4\},\{x_3,x_4\}\}$*, i.e., all two-element subsets of* $X = \{x_1,x_2,x_3,x_4\}$*. We have* $|f| = 6$ *and* $Z(f) = 8$*. In the general case, the ZDD* f *that represents all k-element subsets of a set* $\{x_1,\dots,x_n\}$ *has* $Z(f) = O(kn)$ *nodes, while representing* $|f| = \binom{n}{k}$ *sets.*

Given two ZDDs f and g, the following list of operations is part of what is called a ZDD family algebra. Each operation can be efficiently implemented using ZDDs.

$$f \cup g = \{\alpha \mid \alpha \in f \text{ or } \alpha \in g\} \qquad \textit{union}$$
$$f \cap g = \{\alpha \mid \alpha \in f \text{ and } \alpha \in g\} \qquad \textit{intersection}$$
$$f \setminus g = \{\alpha \mid \alpha \in f \text{ and } \alpha \notin g\} \qquad \textit{difference}$$
$$f \sqcup g = \{\alpha \cup \beta \mid \alpha \in f \text{ and } \beta \in g\} \qquad \textit{join}$$
$$f \sqcap g = \{\alpha \cap \beta \mid \alpha \in f \text{ and } \beta \in g\} \qquad \textit{meet}$$
$$f \searrow g = \{\alpha \in f \mid \beta \in g \text{ implies } \alpha \not\supseteq \beta\} \qquad \textit{nonsupersets}$$

Finally, if f represents the family $\varepsilon_{x'_1} \cup \dots \cup \varepsilon_{x'_l}$ for some subset $\{x'_1,\dots,x'_l\} = X' \subseteq X$, then

$$\binom{f}{k}$$

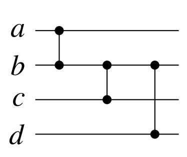
is the ZDD that represents the family $\binom{X'}{k}$.

Note that the *nonsupersets* operation can be described in terms of the others: $f \searrow g = f \setminus (f \sqcup g)$. However, it may be more efficient to implement the operation explicitly in a ZDD package. For a detailed description of how ZDDs are represented in memory and how the ZDD family algebra operations are implemented, the reader is referred to the literature [6].

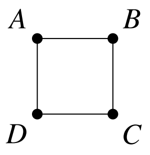
2.3 Physical Quantum Architectures

2.3.1 Rigetti

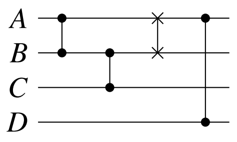
Rigetti has developed quantum machines based on solid-state, superconducting circuit technology. The company has also developed a Python library pyQuil as well as a software development kit (SDK) called *Forest* that can be used to write quantum algorithms, interact with quantum processing units (QPUs) and simulate quantum computing. The quantum instruction language Quil is used to specify algorithms for the Rigetti QPUs [13]. Within the SDK, it is possible to create custom architectures that can be



(a) Circuit



(b) Device



(c) Mapped circuit

Figure 2: Simple circuit and device

targeted by the Rigetti compiler. To implement an algorithm on the Rigetti QPUs, complex gates must be decomposed into the native gate set of $R_z(\theta), R_x(\frac{k\pi}{2})$ where k is an integer value, and the two-qubit operator CZ. As an additional constraint, CZ operations are limited with respect to where they may be placed on the device due to the constraints among qubits. The Rigetti compiler may be used to transform technology-independent circuits into forms that have the appropriate gate library and connections for QPU execution. This compiler performs mapping and minimization procedures, but the resulting circuits may not necessarily be the optimum solution.

2.3.2 IBM

IBM has also developed quantum computers based on solid-state, superconducting circuit technology. The Python SDK *Qiskit* is their tool for performing quantum information processing (QIP) with their platform. Quantum assembly language, or QASM, is used to specify quantum logic for execution on the IBM quantum machines [4]. In order for a quantum circuit to be executable on a real machine, the QASM specification must not only obey coupling map constrictions but also contain operators within the gate set of $R_z(\theta)$, $R_x(\phi)$, $R_x(\gamma)$, and the two-qubit operator CX. The compiler contained in the *Qiskit* SDK may be used for transforming circuits into a technology-ready form, but as with the Rigetti compiler, solutions may not be optimum. *Qiskit* allows for compilation to custom architectures so that devices outside of the existing IBM machines may be targeted.

3 Problem formulation

In this paper, we model the quantum circuit that is to be mapped to a quantum device as a set of pseudo qubits $V = \{v_1, \dots, v_n\}$ and an ordered sequence of two-qubit gates $G = g_1, \dots, g_k$, with $g_i \in \binom{V}{2}$. We can safely ignore the one-qubit gates in the circuit, since the coupling constraints of the device do not affect their mapping. Also note that we do not take the direction of a gate (e.g., the position of control and target in a CX) into account as unidirectional gates may be reversed, as seen in [11], with single qubit operations.

Example 3. *Fig. 2(a) shows a quantum circuit on four pseudo qubits $V = \{a,b,c,d\}$ and three two-qubit gates $g_1 = \{a,b\}$, $g_2 = \{b,c\}$, and $g_3 = \{b,d\}$.*

A quantum device is modeled by an undirected graph (P,E), where $P = \{p_1,\dots,p_m\}$ is a set of physical qubits and an edge $\{p,q\} \in E \subseteq \binom{P}{2}$ states that a 2-qubit operation can be executed using the two physical qubits p and q.

Example 4. *Fig. 2(b) shows a simple four-qubit quantum device with physical qubits $P = \{A,B,C,D\}$ and a ring topology, i.e., edges $E = \{\{A,B\},\{B,C\},\{C,D\},\{D,A\}\}$.*

The goal is to find a mapping $\varphi : V \to P$ of pseudo qubits into physical ones, such that all two-qubit operations in the circuit are executed on adjacent qubits according to the device's coupling constraints. It may not be possible to find such mapping for the input circuit. For example, there exists no such mapping for the example circuit, since the pseudo qubit b interacts with three other qubits. However, by adding SWAP gates to reorder pseudo qubits, a mapping can be achieved. A SWAP gate is a two-qubit operation that can either be implemented with CX or single-qubit rotations and CZ.

Example 5. *By adding a SWAP gate after the second gate of the circuit in Fig. 2(a), one must update all successive gates in order to retain the functionality. However, the transformed circuit can now be mapped to the quantum device by mapping $a \mapsto A$, $b \mapsto B$, $c \mapsto C$, and $d \mapsto D$.*

The aim is to use a small number of SWAP gates when transforming an initial circuit to a circuit that can be mapped into a target device. Finding the globally optimum mapping and a transformed circuit using the fewest number of SWAP gates is a computationally complex and time-consuming task [2].

To address the problem of finding maximal partitions for a circuit using ZDDs, we use partitions where

- there exists a subgraph isomorphism of (V,G_j) to (P,E) for $1 \le j \le l$,
- there exists no subgraph isomorphism of $(V,G_j \cup \{g_{b_{j+1}}\})$ to (P,E) for $1 \le j < l$.

Each partition has an associated set of mappings of pseudo to physical qubits $\Phi_j = \{\varphi : V \mapsto P\}$ where φ is a subgraph isomorphism of (V,G_j) to (P,E). If partition G_j cannot be extended, SWAP operations are inserted to merge the last gate of the partition, g_i with the adjacent gate in the circuit. These swapping operations, referred to as layers, exchange information on the adjacent physical qubits of the device and are executable in parallel within a single time cycle. The best SWAP layer is chosen according to a scoring metric. Once selected, the SWAP layer merges the gate g_i with g_{i+1} by inserting SWAP gates before g_{i+1}, extending G_j.

Ideally, a single partition will cover the entire circuit, providing a set of mappings that assign pseudo qubits to physical qubits on the device. In the case that multiple partitions exist that are maximally sized with inserted SWAP layers, a mapping for the circuit is selected using the largest partition.

4 Finding maximal partitions

In this section, we describe how to use ZDDs and ZDD operations to find a maximal partition that starts in some gate g_i. The ZDDs are defined over the nm variables vp for each $v \in V$ and each $p \in P$. Each ZDD represents a family of finite subsets, and each subset α represents a partial mapping $\varphi : V \to P$, where $\varphi(v) = p$, if and only if $vp \in \alpha$ where α is a mapping.

Example 6. *The ZDD $f = \{\{aA,bB,cC,dD\}\}$ represents a family of a single mapping that maps $a \mapsto A$, $b \mapsto B$, $c \mapsto C$, and $d \mapsto D$.*

First, we define some general sets, which are used throughout the following operations. It is sufficient to initialize these sets once at the beginning of the algorithm. The set

$$\mathit{from}(v) = \bigcup_{p \in P} \varepsilon_{vp} \tag{2}$$

contains all singleton mappings $v \mapsto p$ for some $v \in V$. Analogously, the set

$$\mathit{to}(p) = \bigcup_{v \in V} \varepsilon_{vp} \tag{3}$$

contains all singleton mappings $v \mapsto p$ for some $p \in P$.

Example 7. *For the example circuit and device we have* $\mathit{from}(a) = \{\{aA\},\{aB\},\{aC\},\{aD\}\}$ *and* $\mathit{to}(A) = \{\{aA\},\{bA\},\{cA\},\{dA\}\}$.

Using this set, we can define two other helpful sets, *valid* and *bad*, using the ZDD family algebra operations. The set *valid* contains all two-element partial mappings that are feasible with respect to the coupling constraints of the device:

$$\mathit{valid} = \bigcup_{\{p,q\} \in E} \mathit{to}(p) \sqcup \mathit{to}(q) \tag{4}$$

The set *bad* contains all two element sets of illegal partial mappings, because they either contain an element with two images or two elements which map to the same image:

$$\mathit{bad} = \bigcup_{v \in V} \binom{\mathit{from}(v)}{2} \cup \bigcup_{p \in P} \binom{\mathit{to}(p)}{2} \tag{5}$$

Lemma 1. *$\beta \in \mathit{bad}$ if and only if either $\beta = \{vp, vq\}$ and $p \neq q$ or $\beta = \{pv, pw\}$ and $v \neq w$, for all $v, w \in V$ and all $p, q \in P$.*

Proof. First, note that

$$\mathit{from}(v) \sqcup \mathit{from}(v) = \bigcup_{p \in P} \bigcup_{q \in P} \{\{vp, vq\}\}.$$

From the resulting set, we can remove the cases in which $p = q$ by subtracting $\mathit{from}(v)$. The same argument applies to $(\mathit{to}(p) \sqcup \mathit{to}(p)) \setminus \mathit{to}(p)$. □

Corollary 1. *A set α represents a partial mapping if and only if there exists no $\beta \in \mathit{bad}$ such that $\beta \subseteq \alpha$.*

Last, we define the set $\mathit{map}(i)$ which represents all possible mappings of the pseudo qubits in gate $g_i = \{v, w\}$:

$$\mathit{map}(i) = (\mathit{from}(v) \sqcup \mathit{from}(w)) \cap \mathit{valid} \tag{6}$$

In other words, we first join all possible mappings of v with all possible mappings of w, before we restrict the result two-element subsets to those which are valid with respect to the target device.

Example 8. *Gate $g_1 = \{a, b\}$ can be mapped in eight different ways:*

$$\begin{aligned}\mathit{map}(1) = \{&\{aA, bB\}, \{aB, bC\}, \{aC, bD\}, \{aD, bA\}\\ &\{aB, bA\}, \{aC, bB\}, \{aD, bC\}, \{aA, bD\}\}\end{aligned}$$

Also gate $g_2 = \{b, c\}$ can be mapped in eight different ways:

$$\begin{aligned}\mathit{map}(2) = \{&\{bA, cB\}, \{bB, cC\}, \{bC, cD\}, \{bD, cA\}\\ &\{bB, cA\}, \{bC, cB\}, \{bD, cC\}, \{bA, cD\}\}\end{aligned}$$

```
Data: Gate sequence $g_1, \ldots, g_k$, and device $(P,E)$
Result: partitions $G_j$ with begin and end indexes $b_j, e_j$; all possible mappings $\Phi_j$
Set $j \leftarrow 1$, $b_j \leftarrow 1$, $m \leftarrow map(1)$;
for $i = 2, \ldots, k$ do
    Set $m' \leftarrow (m \sqcup map(i)) \searrow bad$;
    if $m' = \emptyset$ then
        for $layer\ in\ layers$ do
            calculate scores;
        end
        if $max\ score \neq 0$ then
            insert SWAP circuit;
            update topology;
            Set $m(\text{max score})' \leftarrow (m \sqcup map(i)) \searrow bad$;
            Set $m \leftarrow m'$;
        else
            Set $e_j \leftarrow i - 1$, $\Phi_j \leftarrow m$;
            Set $j \leftarrow j + 1$, $b_j \leftarrow i$, $m \leftarrow map(i)$;
        end
    else
        Set $m \leftarrow m'$;
    end
end
Set $e_j \leftarrow k$, $\Phi_j \leftarrow m$;
```

Algorithm 1: Find maximal partitions

Finally, the possible mappings of two consecutive gates g_i and g_{i+1} can be computed using

$$(map(i) \sqcup map(i+1)) \searrow bad. \tag{7}$$

Example 9. *Recall the two families $map(1)$ and $map(2)$ from the previous example. Joining them leads to a family with up to 64 subsets, out of which many do not represent legal partial mappings such as $\{aA, bB, bA, cB\}$. These can be removed using the restriction to bad, resulting a family consisting of the only eight legal partial mappings:*

$$\{\{aA, bB, cC\}, \{aB, bC, cD\}, \{aC, bD, cA\},$$
$$\{aD, bA, cB\}, \{aA, bD, cC\}, \{aB, bA, cD\},$$
$$\{aC, bB, cA\}, \{aD, bC, cB\}\}$$

In some instances, Eqn. 7 results in the empty set, $\emptyset$, whenever the mappings of two consecutive gates are combined. In this case, SWAP procedures must be performed on physical qubits in order to exchange pseudo qubit information and extend the partition G_j. Acceptable SWAP operations are determined by the topology of the device, and the sets of operations that can be executed simultaneously within a time cycle are desired. ZDDs are used to create a set of all "good" SWAP circuits, which are those that interact with at least one qubit in the image of φ and the depth of the circuit is one, i.e., a SWAP circuit may only contain multiple SWAP gates as long as qubits between gates are not shared. The ZDDs for this task differ from those used to enumerate all mappings, and they do not share any variables. The SWAP circuit ZDDs are defined over the $|E|$ variables e for each $e \in E$ since SWAP gates can only be placed on certain edges connecting physical qubits according to the quantum device operational characteristics. We initialize the ZDD base with the following ZDDs. For each $p \in P$, the ZDD

$$edges(p) = \bigcup\{\varepsilon_e \mid e \in E \text{ s.t. } p \in e\} \tag{8}$$

contains all SWAPs that interact with qubit p.

All possible subsets of SWAP gates that can be executed in parallel (i.e., in depth 1) are described by the ZDD

$$layers = \wp \searrow \bigcup_{p \in P} \binom{edges(p)}{2}. \tag{9}$$

Example 10. *The set of SWAP gates that can be parallelized for the topology in Fig. 2(b) can be determined using Eqn. 9. First, the set of edges for all physical qubits p in P must be defined:*

$$edges(A) = \{\{AB\},\{AD\}\},$$

$$edges(B) = \{\{AB\},\{BC\}\},$$

$$edges(C) = \{\{BC\},\{CD\}\},$$

$$edges(D) = \{\{CD\},\{AD\}\}.$$

Next, a set must be created that consists of the union of all of the $\binom{N}{k}$ operations where $N = edges(p)$ and $k = 2$:

$$\bigcup_{p \in P} \binom{edges(p)}{2} = \{\{AB,AD\},\{AB,BC\},\{BC,CD\},\{CD,AD\}\}.$$

Finally, the nonsupersets operation is implemented between the tautology, or universal family of all subsets for the physical qubit edges. The set $\wp$ is defined as

$$\wp = \{\emptyset,\{AB\},\{BC\},\{CD\},\{AD\},\{AB,BC\},\{AB,CD\},\{AB,AD\},\{BC,CD\},\{BC,AD\},$$
$$\{CD,AD\},\{AB,BC,CD\},\{AB,BC,AD\},\{AB,CD,AD\},\{BC,CD,AD\},\{AB,BC,CD,AD\}\},$$

allowing the layers for the device in 2(b) to be calculated as

$$layers = \{\emptyset,\{AB\},\{BC\},\{CD\},\{AD\},\{AB,CD\},\{BC,AD\}\}.$$

Not all combinations of SWAP gates in *layers* may be useful for extending a partition of gates. For example, some SWAP circuits may allow the partition G_j to extend further and have greater depth while others provide more mapping options within Φ_j. As a result, a scoring function is calculated for each member of *layers* to determine the optimal SWAP decision to extend a circuit partition. The function of

$$score = (A\alpha + B\beta)\,\frac{\gamma}{C} \tag{10}$$

is used to select the optimal SWAP circuit where α is depth weight, β is map weight, γ is SWAP weight, A is depth count, B is map count, and C is SWAP count. In Eqn. 10, SWAP count has an inverse relationship with a layer's *score* as lower overall gate counts, or volume, in a technology-mapped implementation are preferred. The parameter γ, however, can be adjusted to make the cost of an additional SWAP operations less severe. The weights of this function can be tuned to prioritize the growth of the partition with respect to either gate coverage or available mappings if a layer in *layers* is implemented. In the case that *score* for each layer is zero, a new partition must be created for the circuit.

Algorithm 1 implements ZDD data structures and the aforementioned ZDD operations to compute maximal partitions of quantum circuits starting from the first gate. A counter j indicates the current partition number as the algorithm parses through the operators in the network. In m, a set of mappings are stored and updated as the current partition increases in size. These maps are eventually stored in Φ_j. For each gate i we try to extend m by adding the gate using $map(i)$, and storing the resulting mappings in m'. If m' is empty, *layers* will be used to determine if a SWAP operation can be implemented in order to increase the current partition. The scores for all of the sets in *layers* are calculated, and the SWAP circuit with the largest score is used to extend the partition. After the SWAP is implemented, the topology of the device is updated, maximum score m' is calculated, and m is updated with m'. If the maximum score is zero, then the then the current partition ends at $i-1$, and a new partition $j+1$ starts at gate i.

5 ZDD mapping in the Quantum Compilation Flow

The algorithm discussed in the previous section implements the mapping of pseudo qubits in a quantum circuit specification to physical qubits on a real device. While the mapping procedure is essential for for quantum compilation, additional optimization steps can further improve the technology-mapped logic. For this reason, we propose the incorporation of the ZDD mapping techniques into a larger logic synthesis flow. In this procedure, mapping would occur after a circuit has been decomposed into one- and two-qubit operators and before a specification is compiled by a device's custom compilation tool. Completing synthesis with available compilation tools allows the opportunity to take advantage of existing optimization algorithms while the operators of the of a circuit are transformed into a platform's native gate library. It should be noted that although the ZDD mapping algorithm was evaluated using superconducting qubits as a target platform, the techniques described here are applicable to other quantum technologies that have coupling restrictions.

6 Experimental Results

The ZDD quantum mapping algorithm was developed in C++ and was incorporated in the *tweedledum* logic synthesis library found in reference [10]. A subset of benchmarks from [1] were selected to evaluate the methods described in this work. These benchmarks contain a variety of arithmetic and quantum algorithms that are originally specified with a gate set that contains physically unrealizable multi-qubit gates. Thus, the specifications are transformed into the Clifford+T library of single- and two-qubit gates using the "phasefold" pass of the *Feynman* toolkit [1] . After this procedure, the benchmarks are in a technology-independent form that can be mapped to a target quantum device.

A ring topology was chosen as the target device during synthesis. Each benchmark was targeted to a device that contained n physical qubits where n is the number of pseudo qubits in a quantum algorithm. In these devices, all qubits are connected to their two adjacent neighbors, as seen in Fig. 2(b), and the connections are bidirectional with respect to the placement of the two-qubit CX gate. Once the circuit and topology are selected, Algorithm 1 is applied to map the pseudo qubits in the design to physical qubits. The scoring operation of Eqn. 10 that chooses between the SWAP circuits in *layers* to extend the partition used zero, one, and one for the depth, map, and SWAP weights, respectively. If the benchmark can be covered by an entire partition during the application of Algorithm 1, then the resulting specification is in a fully technology-mapped form and therefore compatible with the available connections of the target device. If multiple partitions are needed for a benchmark, the resulting specification is mapped using a permutation of the largest partition, making additional SWAP operations required for the design to be fully compatible with the target technology. This additional circuit modification is accomplished by the custom compilers that are provided with the Rigetti and IBM SDKs. Compiling the ZDD mapped circuits into the selected device topology with the available Rigetti and IBM compilers is the final procedure in the mapping flow. This step also ensures that all gates are translated into gates from the target device. Note that such a translation cannot lead to any further violations of the coupling constraints. After the final compilation step, the circuits are ready for execution on their respective platform since they are in a technology-mapped and optimized form. Details about the benchmarks along with experimental results of the mapping and compilation procedures can be found in Table 1.

In Table 1, details about depth, volume, and two-qubit gate count have been included for the ZDD mapped and compiled circuits. Information about circuits that were only ZDD mapped without compilation is also shown. The benchmarks were compiled with and without preprocessing the circuit with the ZDD mapper. Circuits that improved in metrics for a particular device and benchmark whenever ZDD mapping was implemented are emphasized. On average, benchmarks mapped to a Rigetti ring topology saw a decrease of around 10% with respect to depth, volume, and two-qubit gate count whenever ZDD mapping was included in technology-dependent logic synthesis flow before compilation. The IBM-compiled circuits, however, only saw an average decrease of 2.3% in depth, volume, and two-qubit gate count whenever ZDD mapping was used. Individual improvements in circuit metrics of up to approximately a 50% decrease was seen in volume on the Rigetti devices and up to approximately a 44% decrease was seen in depth on the IBM devices. These findings demonstrate the potential that ZDD mapping techniques have with respect to finding more optimal solutions whenever generating technology-dependent forms of quantum circuits.

7 Conclusion

We present a method for mapping quantum logic circuits to actual devices using ZDDs. The required operations and algorithm are described, and the implementation is developed and tested in a quantum compilation flow that targets devices meant for the Rigetti and IBM families of superconducting quantum computers. When experimental results are evaluated, it is observed that incorporating ZDD mapping into quantum logic synthesis in many cases allowed for more optimal circuits to be found in their technology-dependent form. These results suggest that the ZDD may be a useful tool for quantum compilation that should be continued to be investigated in the future.

Acknowledgements. The author thanks Alan Mishchenko for helpful discussion and comments. This work was supported by the Swiss National Science Foundation in the project "Technology-dependent Optimization in Quantum Compilation" (IZSEZ0_184016).

A Implementation of ZDD operations

In [6, Ex. 7.1.4-207], Knuth describes the implementation of a ZDD operation $f \S k$, which is similar to the operation $\binom{f}{k}$, which is used in this paper. No description of an implementation for $\binom{f}{k}$ was found in the literature, and therefore we report our implementation here, in a similar style Knuth used to describe the implementation of $f \S k$.

Table 1: Depth, volume, and two-qubit metrics of benchmarks after zdd mapping, IBM compiling, and Rigetti compiling. Values that decreased whenever ZDD mapping was implemented before compilation have been emphasized.

Benchmark Name	No. Qubits		Original ZDD Mapped	Original Rigetti Compiled	Original IBM Compiled	ZDD Mapped/ Rigetti Compiled	ZDD Mapped/ IBM Compiled
barenco_tof_3	5	*depth:*	64	118	62	98 *(-16.95%)*	84 *(+35.48%)*
		vol.:	95	446	180	221 *(-50.45%)*	165 *(-8.33%)*
		2q gates:	73	68	67	58 *(-14.71%)*	63 *(-5.97%)*
barenco_tof_4	7	*depth:*	94	230	131	155 *(-32.6%)*	130 *(-0.76%)*
		vol:	190	763	462	449 *(-41.15%)*	335 *(-27.49%)*
		2q gates:	152	123	177	117 *(-4.88%)*	132 *(-25.42%)*
barenco_tof_5	9	*depth:*	94	231	121	155 *(-32.9%)*	130 *(+7.44%)*
		vol.:	285	1136	528	682 *(-39.96%)*	505 *(-4.36%)*
		2q gates:	231	184	201	177 *(-3.8%)*	201 *(+0%)*
gf2^4_mult	12	*depth:*	46	337	251	361 *(+7.12%)*	354 *(+41.03%)*
		vol:	232	2319	1450	2593 *(+11.82%)*	1511 *(+4.21%)*
		2q gates:	145	363	557	430 *(+18.46%)*	587 *(+5.39%)*
gf2^5_mult	15	*depth:*	64	422	259	504 *(+19.43%)*	342 *(+32.05%)*
		vol:	363	3747	2212	4510 *(+20.36%)*	2351 *(+6.28%)*
		2q gates:	230	596	842	775 *(+30.03 %)*	910 *(+8.07%)*
grover_5	9	*depth:*	210	968	989	872 *(-9.92%)*	552 *(-44.19%)*
		vol:	777	4857	2909	4484 *(-7.68%)*	2590 *(-10.97%)*
		2q gates:	441	781	1096	739 *(-5.38%)*	1011 *(-7.76%)*
hwb6	7	*depth:*	113	449	269	432 *(-3.79%)*	290 *(+7.81 %)*
		vol:	303	2032	1049	2027 *(-0.25%)*	1101 *(+4.96%)*
		2q gates:	185	332	404	338 *(+1.81%)*	422 *(+4.46%)*
mod_mult _55	9	*depth:*	49	189	123	177 *(-6.35 %)*	144 *(+17.07 %)*
		vol:	155	978	500	850 *(-13.09 %)*	469 *(-6.2%)*
		2q gates:	88	151	193	143 *(-5.3 %)*	176 *(-8.81%)*
mod_5 _4	5	*depth:*	60	115	94	95 *(-17.4%)*	92 *(-2.13%)*
		vol:	121	459	229	308 *(-32.9%)*	239 *(+4.37%)*
		2q gates:	98	73	88	79 *(+8.21%)*	92 *(+4.55%)*
qft _4	5	*depth:*	142	162	155	137 *(-15.43%)*	105 *(-32.26%)*
		vol:	247	447	322	433 *(-3.13%)*	293 *(-9.01%)*
		2q gates:	120	79	126	92 *(+16.46%)*	114 *(-9.52%)*
tof_3	5	*depth:*	39	98	62	72 *(-26.53%)*	61 *(-1.61%)*
		vol.:	75	309	145	195 *(-36.89%)*	135 *(-6.9%)*
		2q gates:	54	47	53	45 *(-4.26%)*	52 *(-1.89%)*
tof _4	7	*depth:*	46	117	98	88 *(-24.79%)*	62 *(-36.73%)*
		vol:	125	505	326	327 *(-35.25%)*	218*(-33.12%)*
		2q gates:	92	80	121	75 *(-6.25%)*	84 *(-30.58%)*
tof _5	9	*depth:*	46	118	68	89 *(-24.58%)*	62 *(-8.82%)*
		vol:	175	707	335	459 *(-35.08%)*	308 *(-8.06%)*
		2q gates:	130	112	132	106 *(-5.36%)*	118 *(-10.61%)*
vbe _adder _3	10	*depth:*	67	216	165	197 *(-8.8%)*	232 *(+40.61%)*
		vol:	162	1244	765	1131 *(-9.08%)*	835 *(+9.15%)*
		2q gates:	122	190	294	195 *(+2.63%)*	329 *(+11.9%)*

CHOOSE(f,k) = “If $k = 1$, return f. If $f = \emptyset$ and $k > 0$, return $\emptyset$. If $f = \emptyset$ and $k = 0$, return $\{\emptyset\}$. If $\binom{f}{k} = r$ is in the cache, return r. Otherwise set $r \leftarrow$ CHOOSE(f_l,k). If $k > 0$, set $q \leftarrow$ CHOOSE$(f_l,k-1)$ and $r \leftarrow$ ZUNIQUE(f_v,r,q). Put $\binom{f}{k} = r$ in the cache, and return r.”

References

[1] Matthew Amy (2019): *Feynman*. Available at `https://github.com/meamy/feynman`.

[2] Adi Botea, Akihiro Kishimoto & Radu Marinescu (2018): *On the Complexity of Quantum Circuit Compilation*. In: Int’l Symp. on Combinatorial Search, pp. 138–142. Available at `https://aaai.org/ocs/index.php/SOCS/SOCS18/paper/view/17959`.

[3] Andrew M Childs, Eddie Schoute & Cem M Unsal (2019): *Circuit Transformations for Quantum Architectures*. arXiv preprint arXiv:1902.09102.

[4] Andrew W Cross, Lev S Bishop, John A Smolin & Jay M Gambetta (2017): *Open quantum assembly language*. arXiv preprint arXiv:1707.03429.

[5] Wakaki Hattori & Shigeru Yamashita (2018): *Quantum Circuit Optimization by Changing the Gate Order for 2D Nearest Neighbor Architectures*. In: Int’l Conf. on Reversible Computation, pp. 228–243, doi:10.1007/978-3-319-99498-7_16. Available at `https://doi.org/10.1007/978-3-319-99498-7_16`.

[6] Donald Ervin Knuth (2011): *The Art of Computer Programming, Volume 4A*. Addison-Wesley.

[7] Gushu Li, Yufei Ding & Yuan Xie (2018): *Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices*. arXiv preprint arXiv:1809.02573.

[8] Shin-ichi Minato (1993): *Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems*. In: Design Automation Conference, pp. 272–277, doi:10.1145/157485.164890. Available at `https://doi.org/10.1145/157485.164890`.

[9] John Preskill (2018): *Quantum Computing in the NISQ era and beyond*. Quantum 2, p. 79. ArXiv preprint arXiv:1801.00862v3.

[10] Bruno Schmitt et al. (2019): *Tweedledum*. Available at `https://github.com/boschmitt/tweedledum`.

[11] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Sylvain Collange & Fernando Magno Quintão Pereira (2018): *Qubit allocation*. In: Int’l Symp. on Code Generation and Optimization, pp. 113–125, doi:10.1145/3168822. Available at `http://doi.acm.org/10.1145/3168822`.

[12] Kaitlin N. Smith & Mitchell A. Thornton (2018): *Automated Mapping Methods for the IBM Transmon Devices*. In: International Workshop on Post-Binary ULSI Systems (ULSI-WS), pp. 12–17. Available at `https://s2.smu.edu/~mitch/ftp_dir/pubs/ulsiws18.pdf`.

[13] Robert S Smith, Michael J Curtis & William J Zeng (2016): *A practical quantum instruction set architecture*. arXiv preprint arXiv:1608.03355.

[14] Alwin Zulehner, Alexandru Paler & Robert Wille (2018): *An efficient methodology for mapping quantum circuits to the IBM QX architectures*. arXiv preprint arXiv:1712.04722v3.