Reducing the CNOT Count for Clifford+T Circuits on NISQ Architectures

Vlad Gheorghiu[2], Sarah Meng Li[1], Michele Mosca[2], and Priyanka Mukhopadhyay[2].

[1]Department of Mathematics and Statistics, Dalhousie University, Halifax NS, Canada
[2]Institute for Quantum Computing, University of Waterloo, Waterloo ON, Canada

June 22nd 2021

Background

Compilation: Translating a program to a set of elementary quantum gates.

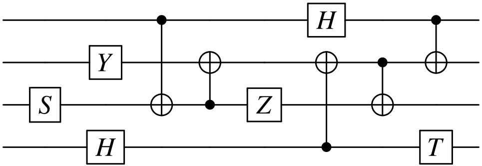
Implementation: Mapping unitary operations to physical architectures.

Connectivity constraint: Applying a multi-qubit gate on admissible qubits.

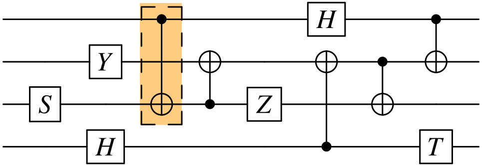
Clifford+T Circuits

Clifford+T circuits are quantum circuits over the gate set

$$\{CNOT, H, T, S, X, Y, Z\}.$$



Basic Gates



- CNOT acts on two qubits, *control* c and *target* t :

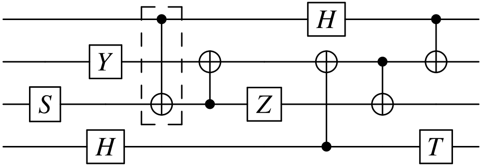
$$CNOT\,|c,t\rangle = |c, c \oplus t\rangle\,.$$

- $X,\ Y,\ Z,\ T,\ S$ act on a single qubit:

$$X\,|t\rangle = |t \oplus 1\rangle\,,\ Y\,|t\rangle = \omega^{4t}\,|t \oplus 1\rangle\,,\ Z\,|t\rangle = \omega^{4t}\,|t\rangle\,,\ S\,|t\rangle = \omega^{2t}\,|t\rangle\,,\ T\,|t\rangle = \omega^{t}\,|t\rangle\,,$$

$c, t \in \mathbb{F}_2,\ \omega = e^{\frac{i\pi}{4}},\ \oplus$ corresponds to Boolean exclusive-OR.

Basic Gates



- CNOT acts on two qubits, *control* c and *target* t :

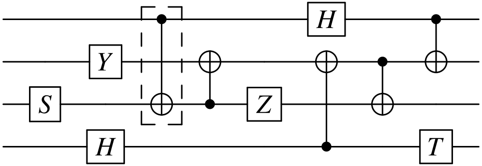
$$CNOT\,|c,t\rangle = |c, c\oplus t\rangle\,.$$

- $X,\ Y,\ Z,\ T,\ S$ act on a single qubit:

$$X\,|t\rangle = |t\oplus 1\rangle\,,\ Y\,|t\rangle = \omega^{4t}\,|t\oplus 1\rangle\,,\ Z\,|t\rangle = \omega^{4t}\,|t\rangle\,,\ S\,|t\rangle = \omega^{2t}\,|t\rangle\,,\ T\,|t\rangle = \omega^{t}\,|t\rangle\,,$$

$c, t\in\mathbb{F}_2,\ \omega = e^{\frac{i\pi}{4}},\ \oplus$ corresponds to Boolean exclusive-OR.

Basic Gates



- CNOT acts on two qubits, *control* c and *target* t :

$$CNOT\,|c,t\rangle = |c, c \oplus t\rangle\,.$$

- $X,\ Y,\ Z,\ T,\ S$ act on a single qubit:

$$X\,|t\rangle = |t \oplus 1\rangle\,,\ Y\,|t\rangle = \omega^{4t}\,|t \oplus 1\rangle\,,\ Z\,|t\rangle = \omega^{4t}\,|t\rangle\,,\ S\,|t\rangle = \omega^{2t}\,|t\rangle\,,\ T\,|t\rangle = \omega^{t}\,|t\rangle\,,$$

$c, t \in \mathbb{F}_2,\ \omega = e^{\frac{i\pi}{4}},\ \oplus$ corresponds to Boolean exclusive-OR.

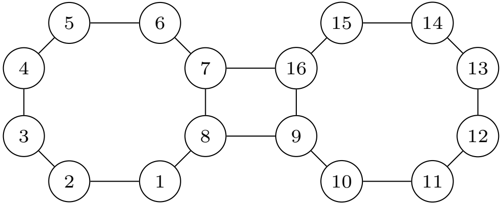
Connectivity Graph

Definition

A ***graph*** is a pair $G = (V_G, E_G)$ where V_G is a set of *vertices* and E_G is a set of pairs $e = (u, v)$ such that $u, v \in V_G$. Each such pair is called an *edge*.

Remark: We are interested in the *simple undirected connected graphs*.

- Simple: there is at most one edge between two distinct vertices and no self-loops.
- Undirected: edges have no direction.

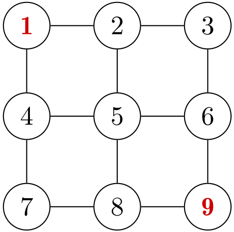


Rigetti 16Q-Aspen

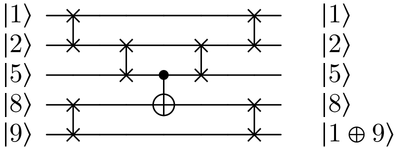
Naive Solution

Naively we can insert SWAP operators to move a pair of logical qubits to physical positions admissible for two-qubit operations.

Example Performing $CNOT_{1,9}$ under the given connectivity constraint.



9-Qubit Square Grid

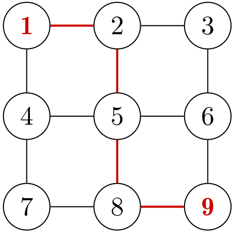


$CNOT_{1,9}$ with SWAPs

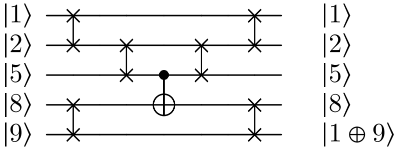
Naive Solution

Naively we can insert SWAP operators to move a pair of logical qubits to physical positions admissible for two-qubit operations.

Example Performing $CNOT_{1,9}$ under the given connectivity constraint.

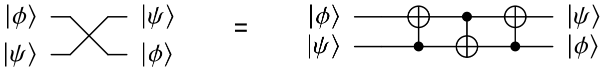


9-Qubit Square Grid



$CNOT_{1,9}$ with SWAPs

Motivation



- If the shortest path length between vertices corresponding to c and t in G is ℓ, the naive solution requires about $6(\ell - 1)$ CNOT gates.

- This entails a significant increase in CNOT-count.

- **Can we reduce the CNOT-count while respecting the connectivity constraint?**

Related Work

We were inspired to use the following techniques.

- Steiner tree problem reduction[1,2].

[1] Beatrice Nash, Vlad Gheorghiu, and Michele Mosca. "Quantum circuit optimizations for NISQ architectures". In: *Quantum Science and Technology* 5.2 (2020), p. 025010.

[2] Aleks Kissinger and Arianne Meijer-van de Griend. "CNOT circuit extraction for topologically-constrained quantum memories". In: *arXiv preprint arXiv:1904.00633* (2019).

[3] Ketan N Patel, Igor L Markov, and John P Hayes. "Optimal synthesis of linear reversible circuits". In: *Quantum Information & Computation* 8.3 (2008), pp. 282–294.

[4] Matthew Amy, Parsiad Azimzadeh, and Michele Mosca. "On the controlled-NOT complexity of controlled-NOT–phase circuits". In: *Quantum Science and Technology* 4.1 (2018), p. 015002.

Related Work

We were inspired to use the following techniques.

- Steiner tree problem reduction[1,2].
- Linear reversible circuits synthesis[3].
- Parity network synthesis[4].

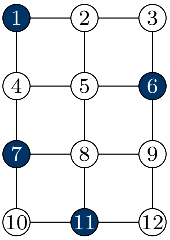
[1] Nash, Gheorghiu, and Mosca, "Quantum circuit optimizations for NISQ architectures".
[2] Kissinger and Griend, "CNOT circuit extraction for topologically-constrained quantum memories".
[3] Patel, Markov, and Hayes, "Optimal synthesis of linear reversible circuits".
[4] Amy, Azimzadeh, and Mosca, "On the controlled-NOT complexity of controlled-NOT–phase circuits".

Steiner Tree

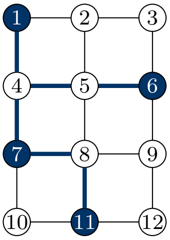
Definition

Given a graph $G = (V_G, E_G)$ and a set of vertices $\mathcal{S} \subseteq V_G$, a ***Steiner tree*** $T = (V_T, E_T)$ is a minimum spanning tree such that $\mathcal{S} \subseteq V_T$.

Example G is a simple undirected graph.



Terminals: $\mathcal{S} = \{1, 6, 7, 11\}$.



A solution to the Steiner tree problem on G.
Steiner nodes: $V_T \setminus \mathcal{S} = \{4, 5, 8\}$

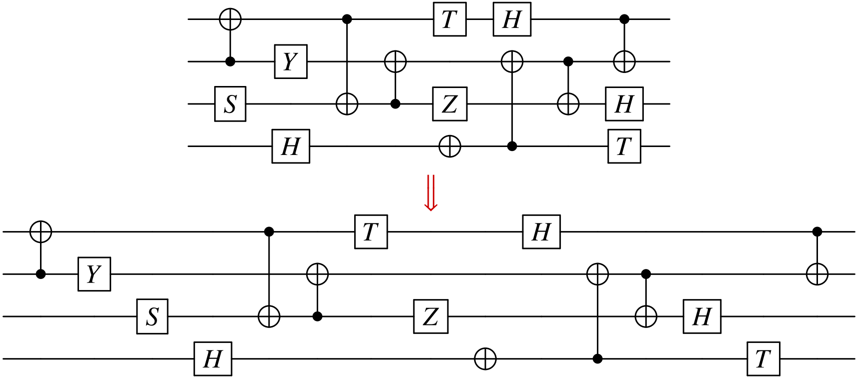
Slice-and-Build Technique[5]

Slice: Partition the circuit based on the locality of H gates.

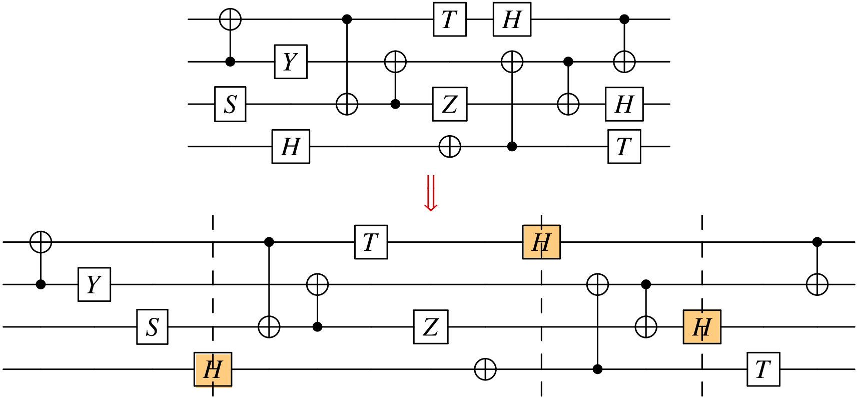
Build: Re-synthesize the sliced portions so that connectivity is respected and the CNOT count is reduced.

[5] Vlad Gheorghiu et al. "Reducing the CNOT count for Clifford+ T circuits on NISQ architectures". In: *arXiv preprint arXiv:2011.12191* (2020).

Slice



Slice



Build

- Each subcircuit is composed of $\mathcal{G}_{ph} = \{CNOT, T, T^\dagger, S, S^\dagger, X, Y, Z\}$.
- Calculate the phase polynomial $\mathcal{P}$ and overall linear transformation $\mathbf{A}_{slice}$.

Build

- Each subcircuit is composed of $\mathcal{G}_{ph} = \{CNOT, T, T^\dagger, S, S^\dagger, X, Y, Z\}$.
- Calculate the phase polynomial $\mathcal{P}$ and overall linear transformation $\mathbf{A}_{slice}$.
- Synthesize a phase polynomial network $\mathcal{C}_{ph}$ over $\mathcal{G}_{ph}$.

Phase Polynomial Network Synthesis

Build

- Each subcircuit is composed of $\mathcal{G}_{ph} = \{CNOT, T, T^\dagger, S, S^\dagger, X, Y, Z\}$.
- Calculate the phase polynomial $\mathcal{P}$ and overall linear transformation $\mathbf{A}_{slice}$.
- Synthesize a phase polynomial network C_{ph} over $\mathcal{G}_{ph}$.

  Phase Polynomial Network Synthesis

- Calculate the overall linear transformation $\mathbf{A}_{ph}$ of C_{ph}.

Build

- Each subcircuit is composed of $\mathcal{G}_{ph} = \{CNOT, T, T^\dagger, S, S^\dagger, X, Y, Z\}$.
- Calculate the phase polynomial $\mathcal{P}$ and overall linear transformation $\mathbf{A}_{slice}$.
- Synthesize a phase polynomial network C_{ph} over $\mathcal{G}_{ph}$.

  Phase Polynomial Network Synthesis

- Calculate the overall linear transformation $\mathbf{A}_{ph}$ of C_{ph}.
- Derive the residual linear transformation $\mathbf{A} = \mathbf{A}_{ph}^{-1}\mathbf{A}_{slice}$.

Build

- Each subcircuit is composed of $\mathcal{G}_{ph} = \{CNOT, T, T^\dagger, S, S^\dagger, X, Y, Z\}$.
- Calculate the phase polynomial $\mathcal{P}$ and overall linear transformation $\mathbf{A}_{slice}$.
- Synthesize a phase polynomial network C_{ph} over $\mathcal{G}_{ph}$.

  Phase Polynomial Network Synthesis

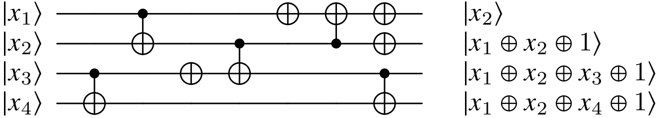
- Calculate the overall linear transformation $\mathbf{A}_{ph}$ of C_{ph}.
- Derive the residual linear transformation $\mathbf{A} = \mathbf{A}_{ph}^{-1}\mathbf{A}_{slice}$.
- Synthesize a linear reversible circuit C_{lin} over $\{CNOT, X\}$.

  Linear Transformation Synthesis

Synthesize Circuits over $\{CNOT, X\}$

For an n–qubit circuit over $\{CNOT, X\}$, we represent the overall linear transformation using an $n \times (n+1)$ binary matrix.

Example

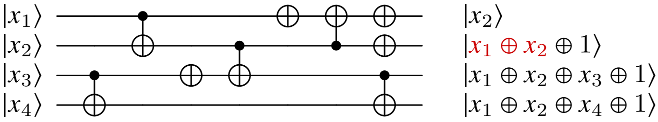


$$A = \begin{array}{c} \begin{array}{ccccc} x_1 & x_2 & x_3 & x_4 & b \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

Synthesize Circuits over $\{CNOT, X\}$

For an n–qubit circuit over $\{CNOT, X\}$, we represent the overall linear transformation using an $n \times (n+1)$ binary matrix.

Example

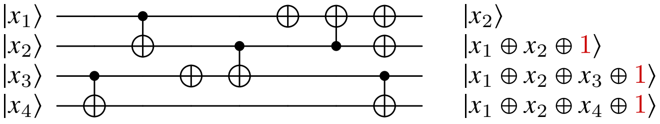


$$
A = \begin{array}{c}
\begin{array}{ccccc} x_1 & x_2 & x_3 & x_4 & b \end{array} \\
\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix}
\end{array}
$$

Synthesize Circuits over $\{CNOT, X\}$

For an n–qubit circuit over $\{CNOT, X\}$, we represent the overall linear transformation using an $n \times (n+1)$ binary matrix.

Example



$$A = \begin{array}{c} \begin{array}{ccccc} x_1 & x_2 & x_3 & x_4 & b \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

Linear Transformation Synthesis

Reverse Engineering

(a) Make $\mathbf{b} = \mathbf{0}$ by applying X to corresponding qubits.

(b) Carry out an analogue of Gaussian elimination.

(c) Use Steiner tree to incorporate connectivity constraints.

Linear Transformation Synthesis

Reverse Engineering

(a) Make $\mathbf{b} = \mathbf{0}$ by applying X to corresponding qubits.

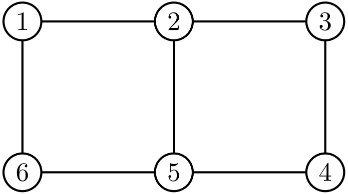
(b) Carry out an analogue of Gaussian elimination.

(c) Use Steiner tree to incorporate connectivity constraints.

Example Let A be a linear transformation and G be the connectivity graph.

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$G =$



Step 1: Reducing A to Upper Triangular Form

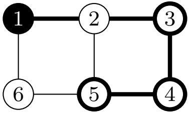
Row Operations I

(a) Starting from the left most column, fix one column at a time.

(b) Fixing the ith column means applying row operations such that $A_{ii} = 1$ and $A_{ji} = 0$ for every $j > i$.

$$A = \begin{bmatrix} \mathbf{1} & 1 & 0 & 1 & 1 & 0 \\ \mathbf{0} & 0 & 1 & 1 & 0 & 1 \\ \mathbf{1} & 0 & 1 & 0 & 1 & 0 \\ \mathbf{1} & 1 & 0 & 1 & 0 & 0 \\ \mathbf{1} & 1 & 1 & 1 & 0 & 0 \\ \mathbf{0} & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$T_{1,\{1,3,4,5\}} =$



$T_1 =$

$T_2 =$

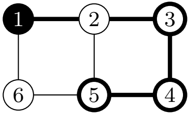
$T_3 =$

- The Steiner tree $T_{1,\mathcal{S}}$ with pivot 1 and terminals $\mathcal{S} = \{1, 3, 4, 5\}$.
- Invoke a sequence of row operations starting from the last sub-tree T_3.

Step 1: Reducing A to Upper Triangular Form

$$A = \begin{bmatrix} \mathbf{1} & 1 & 0 & 1 & 1 & 0 \\ \mathbf{0} & 0 & 1 & 1 & 0 & 1 \\ \mathbf{1} & 0 & 1 & 0 & 1 & 0 \\ \mathbf{1} & 1 & 0 & 1 & 0 & 0 \\ \mathbf{1} & 1 & 1 & 1 & 0 & 0 \\ \mathbf{0} & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

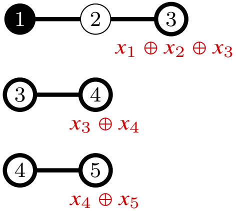
$T_{1,\{1,3,4,5\}} =$



$T_1 =$

$T_2 =$

$T_3 =$



- Propagate 1 from the root to cancel the 1 at the leaf.
- Cancel the 1s in the intermediate Steiner nodes.
- After traversing subtrees and concatenating CNOTs,

$$\mathcal{Y}_1 = \text{CNOT}_{45}\text{CNOT}_{34}\text{CNOT}_{12}\text{CNOT}_{23}\text{CNOT}_{12}.$$

Step 1: Reducing A to Upper Triangular Form

- When $\text{CNOT}_{j,i}$ is applied, row j is added to row i mod 2, and row j remains unchanged.
- After a series of row operations, A is reduced to an upper triangular form.

$$A = \begin{bmatrix} \mathbf{1} & 1 & 0 & 1 & 1 & 0 \\ \mathbf{0} & 0 & 1 & 1 & 0 & 1 \\ \mathbf{1} & 0 & 1 & 0 & 1 & 0 \\ \mathbf{1} & 1 & 0 & 1 & 0 & 0 \\ \mathbf{1} & 1 & 1 & 1 & 0 & 0 \\ \mathbf{0} & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \xrightarrow{\mathcal{Y}_1} A = \begin{bmatrix} \mathbf{1} & 1 & 0 & 1 & 1 & 0 \\ \mathbf{0} & 0 & 1 & 1 & 0 & 1 \\ \mathbf{0} & 1 & 0 & 0 & 0 & 1 \\ \mathbf{0} & 1 & 1 & 1 & 1 & 0 \\ \mathbf{0} & 0 & 1 & 0 & 0 & 0 \\ \mathbf{0} & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \xrightarrow{\mathcal{Y}_2} \cdots \xrightarrow{\mathcal{Y}_6} A = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 2: Reducing $A^\top$ to Identity

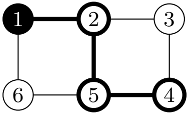
Row Operations II

(a) Starting from the left most column, fix one column at a time.

(b) Fixing the ith column means applying row operations such that $A_{ii} = 1$ and $A_{ji} = 0$ for every $j > i$.

$$A = \begin{bmatrix} \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & 1 & 0 & 0 & 0 & 0 \\ \mathbf{0} & 1 & 1 & 0 & 0 & 0 \\ \mathbf{1} & 1 & 1 & 1 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 1 & 0 \\ \mathbf{0} & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

$T_{1,\{1,2,4,5\}} =$



$T_1 =$

$x_1 \oplus x_2$

$T_2 =$

$x_2 \oplus x_5$

$T_3 =$

$x_5 \oplus x_4$

- A row should be XORed with a row of lower index.
- If not, apply a correction procedure after traversing all sub-trees.

Synthesize Circuits over $\{CNOT, X, T\}$

- Consider circuits over the gate set

$$\mathcal{G}_{ph} = \{CNOT, X, T, T^\dagger, S, S^\dagger, Y, Z\}.$$

- $CNOT\,|x, y\rangle = |x, x \oplus y\rangle$, $T\,|x\rangle = \omega^x\,|x\rangle$, where $\omega = e^{\frac{i\pi}{4}}$ and $x, y \in \mathbb{F}_2$.

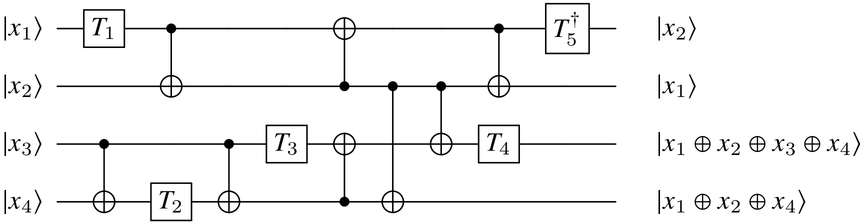
Synthesize Circuits over $\{CNOT, X, T\}$

- Consider circuits over the gate set

$$\mathcal{G}_{ph} = \{CNOT, X, T, T^\dagger, S, S^\dagger, Y, Z\}.$$

- $CNOT\,|x, y\rangle = |x, x \oplus y\rangle$, $T\,|x\rangle = \omega^x\,|x\rangle$, where $\omega = e^{\frac{i\pi}{4}}$ and $x, y \in \mathbb{F}_2$.

Example



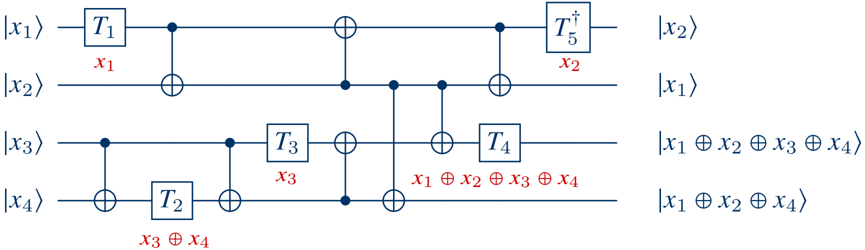
Synthesize Circuits over $\{CNOT, X, T\}$

- Consider circuits over the gate set

$$\mathcal{G}_{ph} = \{CNOT, X, T, T^\dagger, S, S^\dagger, Y, Z\}.$$

- $CNOT\,|x, y\rangle = |x, x \oplus y\rangle$, $T\,|x\rangle = \omega^x\,|x\rangle$, where $\omega = e^{\frac{i\pi}{4}}$ and $x, y \in \mathbb{F}_2$.

Example



$T_1 : \omega^{x_1}$

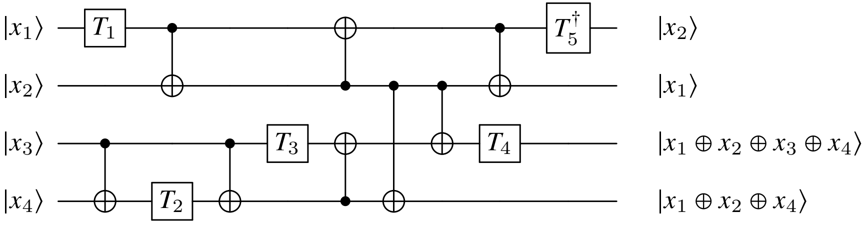
$T_2 : \omega^{x_3 \oplus x_4}$

$T_3 : \omega^{x_3}$

$T_4 : \omega^{x_1 \oplus x_2 \oplus x_3 \oplus x_4}$

$T_5 : \omega^{7x_2}$

Phase Polynomial Network [Amy et al., 2014]



$T_1 : \omega^{x_1}$

$T_2 : \omega^{x_3 \oplus x_4}$

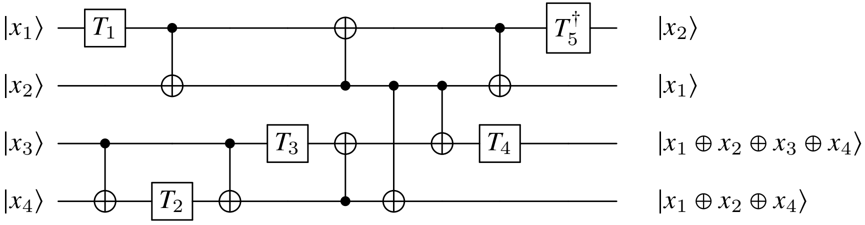
$T_3 : \omega^{x_3}$

$T_4 : \omega^{x_1 \oplus x_2 \oplus x_3 \oplus x_4}$

$T_5 : \omega^{7x_2}$

- A ***representation*** of linear reversible functions: $\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$.

Phase Polynomial Network [Amy et al., 2014]



$T_1 : \omega^{x_1}$

$T_2 : \omega^{x_3 \oplus x_4}$

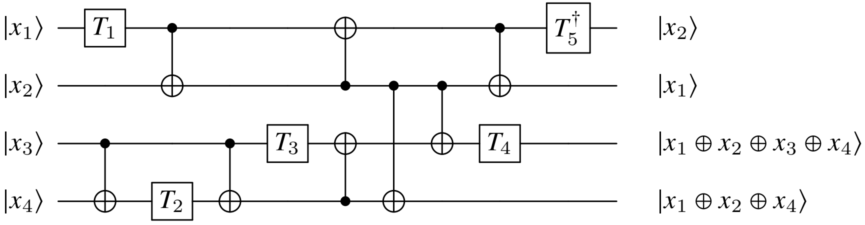
$T_3 : \omega^{x_3}$

$T_4 : \omega^{x_1 \oplus x_2 \oplus x_3 \oplus x_4}$

$T_5 : \omega^{7x_2}$

- A ***representation*** of linear reversible functions: $\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$.
- A ***phase polynomial set***: $\mathcal{P} = \{(1, x_1), (1, x_3 \oplus x_4), (1, x_3), (1, x_1 \oplus x_2 \oplus x_3 \oplus x_4), (7, x_2)\}$.

Phase Polynomial Network [Amy et al., 2014]



$T_1 : \omega^{x_1}$

$T_2 : \omega^{x_3 \oplus x_4}$

$T_3 : \omega^{x_3}$

$T_4 : \omega^{x_1 \oplus x_2 \oplus x_3 \oplus x_4}$

$T_5 : \omega^{7x_2}$

- A ***representation*** of linear reversible functions: $\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}.$
- A ***phase polynomial set***: $\mathcal{P} = \{(1, x_1), (1, x_3 \oplus x_4), (1, x_3), (1, x_1 \oplus x_2 \oplus x_3 \oplus x_4), (7, x_2)\}.$
- A ***phase polynomial network***: a 4-qubit circuit over $\{CNOT, X, T\}$ such that for every $(c, f) \in \mathcal{P}$, f appears before a gate in $\{T, T^\dagger, S, S^\dagger, Y, Z\}$.

Phase Polynomial Network Synthesis

(a) Calculate a parity network matrix M to represent $\mathcal{P}$.

(b) Construct Steiner tress to impose connectivity constraints.

(c) Synthesize a circuit over $\{CNOT, X\}$ that realizes each column in M.

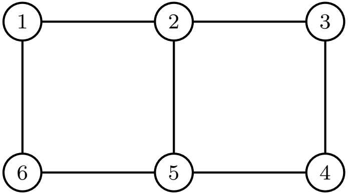
(d) Apply $\{T, T^{\dagger}, S, S^{\dagger}, Y, Z\}$ depending on the coefficients of the parity terms $(c, f) \in \mathcal{P}$.

Columns Represent Parity Term

$$\mathcal{P} = \{(1, 1 \oplus x_1 \oplus x_4 \oplus x_5), (2, x_2 \oplus x_3 \oplus x_5 \oplus x_6), (4, 1 \oplus x_4 \oplus x_5 \oplus x_6), (4, 1 \oplus x_1 \oplus x_2 \oplus x_6),$$
$$(6, 1 \oplus x_1 \oplus x_2 \oplus x_3), (7, 1 \oplus x_1 \oplus x_2 \oplus x_4 \oplus x_6), (1, x_2 \oplus x_4 \oplus x_5)\}$$

$$M = \begin{array}{c} \begin{array}{ccccccc} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 2 & 4 & 4 & 6 & 7 & 1 \end{bmatrix} \end{array}$$

$G =$



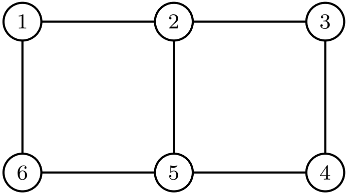
The parity matrix $M_{8\times7}$ and connectivity graph G.

Top Six Rows Encode Parity

$$\mathcal{P} = \{(1, 1 \oplus x_1 \oplus x_4 \oplus x_5), (2, x_2 \oplus x_3 \oplus x_5 \oplus x_6), (4, 1 \oplus x_4 \oplus x_5 \oplus x_6), (4, 1 \oplus x_1 \oplus x_2 \oplus x_6),$$
$$(6, 1 \oplus x_1 \oplus x_2 \oplus x_3), (7, 1 \oplus x_1 \oplus x_2 \oplus x_4 \oplus x_6), (1, x_2 \oplus x_4 \oplus x_5)\}$$

$$M = \begin{array}{c} \begin{array}{ccccccc} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 2 & 4 & 4 & 6 & 7 & 1 \end{bmatrix} \end{array}$$

$G =$



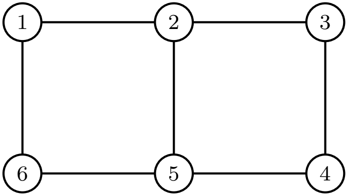
The parity matrix $M_{8\times 7}$ and connectivity graph G.

The 7th Row Encodes Bit Flip

$$\begin{aligned}\mathcal{P} \quad = \quad & \{(1, \textcolor{red}{1} \oplus x_1 \oplus x_4 \oplus x_5), (2, x_2 \oplus x_3 \oplus x_5 \oplus x_6), (4, \textcolor{red}{1} \oplus x_4 \oplus x_5 \oplus x_6), (4, \textcolor{red}{1} \oplus x_1 \oplus x_2 \oplus x_6), \\ & (6, \textcolor{red}{1} \oplus x_1 \oplus x_2 \oplus x_3), (7, \textcolor{red}{1} \oplus x_1 \oplus x_2 \oplus x_4 \oplus x_6), (\textcolor{red}{1}, x_2 \oplus x_4 \oplus x_5)\}\end{aligned}$$

$$M = \begin{array}{c} \begin{array}{ccccccc} \textcolor{green}{\underline{p_1}} & \textcolor{green}{\underline{p_2}} & \textcolor{green}{\underline{p_3}} & \textcolor{green}{\underline{p_4}} & \textcolor{green}{\underline{p_5}} & \textcolor{green}{\underline{p_6}} & \textcolor{green}{\underline{p_7}} \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ \textcolor{red}{1} & \textcolor{red}{0} & \textcolor{red}{1} & \textcolor{red}{1} & \textcolor{red}{1} & \textcolor{red}{1} & \textcolor{red}{0} \\ 1 & 2 & 4 & 4 & 6 & 7 & 1 \end{bmatrix} \end{array}$$

$G =$



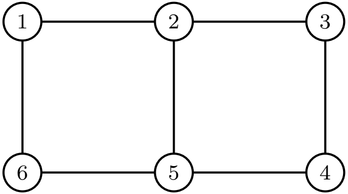
The parity matrix $M_{8\times 7}$ and connectivity graph G.

The Last Row Stores Coefficients

$$\mathcal{P} = \{(1, 1 \oplus x_1 \oplus x_4 \oplus x_5), (2, x_2 \oplus x_3 \oplus x_5 \oplus x_6), (4, 1 \oplus x_4 \oplus x_5 \oplus x_6), (4, 1 \oplus x_1 \oplus x_2 \oplus x_6), (6, 1 \oplus x_1 \oplus x_2 \oplus x_3), (7, 1 \oplus x_1 \oplus x_2 \oplus x_4 \oplus x_6), (1, x_2 \oplus x_4 \oplus x_5)\}$$

$$M = \begin{array}{c} \begin{array}{ccccccc} \underline{p_1} & \underline{p_2} & \underline{p_3} & \underline{p_4} & \underline{p_5} & \underline{p_6} & \underline{p_7} \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 2 & 4 & 4 & 6 & 7 & 1 \end{bmatrix} \end{array}$$

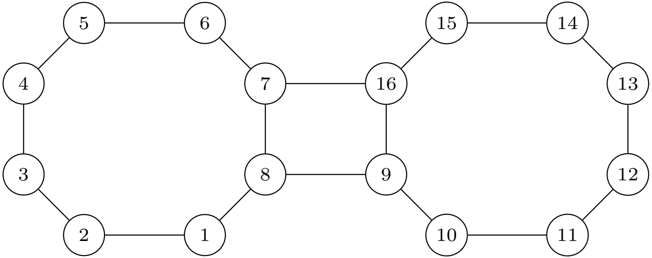
$G =$



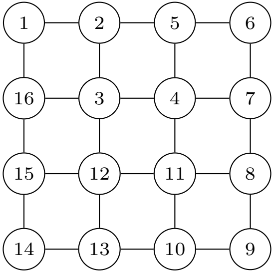
The parity matrix $M_{8\times 7}$ and connectivity graph G.

Implementation

We simulated benchmarks and random circuits on five popular architectures such as (1) 9-qubit square grid, (2) Rigetti 16-qubit Aspen, (3) 16-qubit square grid, (4) 16-qubit IBM QX5, and (5) 20-qubit IBM Tokyo.



(2) Rigetti 16Q-Aspen



(3) 16q-Square Grid

Compare the Increase in CNOT-Count

Architecture	#Qubits	Initial count	SWAP-template Count	Slide-and-Build Count	Slide-and-Build Time
9q-square	9	3	560%	0%	0.184s
		5	612%	146%	0.146s
		10	594%	105%	0.167s
		20	546%	176%	0.2s
		30	596%	185%	0.233s
16q-square	16	4	1050%	238%	0.23s
		8	840%	146%	0.27s
		16	818%	158%	0.43s
		32	853%	341%	0.41s
		64	893%	221%	0.49s
		128	859%	211%	0.57s
		256	897%	238%	0.72s
rigetti-16q-aspen	16	4	1680%	355%	0.23s
		8	1740%	253%	0.396s
		16	1620%	351%	0.47s
		32	1794%	470%	0.48s
		64	1755%	399%	0.66s
		128	1761%	368%	0.58s
		256	1757%	411%	0.61s

Compare the Increase in CNOT Count

Architecture	#Qubits	Initial count	SWAP-template Count	Slide-and-Build	
				Count	Time
9q-square	9	3	560%	0%	0.184s
		5	612%	146%	0.146s
		10	594%	105%	0.167s
		20	546%	176%	0.2s
		30	596%	185%	0.233s
16q-square	16	4	1050%	238%	0.23s
		8	840%	146%	0.27s
		16	818%	158%	0.43s
		32	853%	341%	0.41s
		64	893%	221%	0.49s
		128	859%	211%	0.57s
		256	897%	238%	0.72s
rigetti-16q-aspen	16	4	1680%	355%	0.23s
		8	1740%	253%	0.396s
		16	1620%	351%	0.47s
		32	1794%	470%	0.48s
		64	1755%	399%	0.66s
		128	1761%	368%	0.58s
		256	1757%	411%	0.61s

Compare the Increase in CNOT Count

Architecture	#Qubits	Initial count	SWAP-template Count	Slide-and-Build Count	Slide-and-Build Time
ibm-qx5	16	4	1260%	173%	0.38s
		8	1035%	295%	0.36s
		16	1043%	283%	0.41s
		32	1179%	398%	0.42s
		64	1131%	339%	0.45s
		128	1111%	345%	0.575s
		256	1141%	380%	0.73s
ibm-q20-tokyo	20	4	525%	128%	0.186s
		8	555%	275%	0.295s
		16	570%	88%	0.37s
		32	501%	154%	0.55s
		64	543%	137%	0.54s
		128	540%	141%	0.645s
		256	535%	125%	0.72s

Conclusion

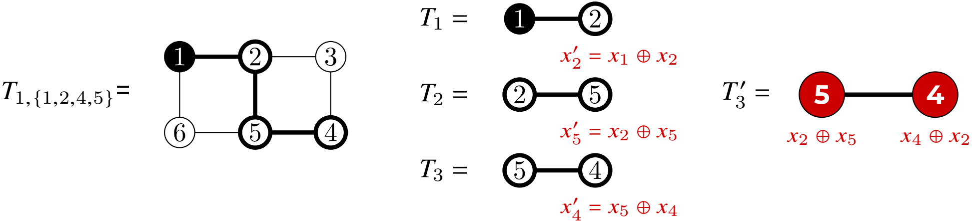
- We designed a heuristic algorithm that reduces the CNOT count in Clifford+T circuits while accounting for the connectivity constraints.
- It’s possible to improve the results by optimizing the initial mapping from logical qubits to physical qubits.
- Moving forward, we would like to rigorously benchmark against other compilers such as tket by CQC and Qiskit by IBM.

Thank you!

Avoid Disturbing 0s' in Upper Triangle

- A node (row) should be XORed with a row with a higher-index.
- If not, apply a correction procedure after traversing all sub-trees.

Example Work with a violation in T_3.



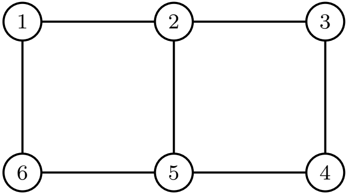
- Take the shortest path from 5 to 4 and apply the same traversals: CNOT_{54}
- The parity at 4 becomes $x'_4 \oplus x'_5 = (x_5 \oplus x_4) \oplus (x_2 \oplus x_5) = x_4 \oplus x_2$.

Columns Represent Parity Term

$$\begin{aligned}\mathcal{P} \quad = \quad & \{(1, 1 \oplus x_1 \oplus x_4 \oplus x_5), (2, x_2 \oplus x_3 \oplus x_5 \oplus x_6), (4, 1 \oplus x_4 \oplus x_5 \oplus x_6), (4, 1 \oplus x_1 \oplus x_2 \oplus x_6), \\ & (6, 1 \oplus x_1 \oplus x_2 \oplus x_3), (7, 1 \oplus x_1 \oplus x_2 \oplus x_4 \oplus x_6), (1, x_2 \oplus x_4 \oplus x_5)\}\end{aligned}$$

$$P = \begin{array}{c} \begin{array}{ccccccc} \underline{p_1} & \underline{p_2} & \underline{p_3} & \underline{p_4} & \underline{p_5} & \underline{p_6} & \underline{p_7} \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 2 & 4 & 4 & 6 & 7 & 1 \end{bmatrix} \end{array}$$

$G =$



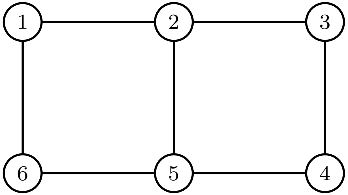
The parity matrix $P_{8\times 7}$ and connectivity graph G.

Top Six Rows Encode Parity

$$\begin{aligned}\mathcal{P} \quad = \quad & \{(1, 1 \oplus x_1 \oplus x_4 \oplus x_5), (2, x_2 \oplus x_3 \oplus x_5 \oplus x_6), (4, 1 \oplus x_4 \oplus x_5 \oplus x_6), (4, 1 \oplus x_1 \oplus x_2 \oplus x_6), \\ & (6, 1 \oplus x_1 \oplus x_2 \oplus x_3), (7, 1 \oplus x_1 \oplus x_2 \oplus x_4 \oplus x_6), (1, x_2 \oplus x_4 \oplus x_5)\}\end{aligned}$$

$$P = \begin{array}{c} \begin{matrix} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 2 & 4 & 4 & 6 & 7 & 1 \end{bmatrix} \end{array}$$

$G =$



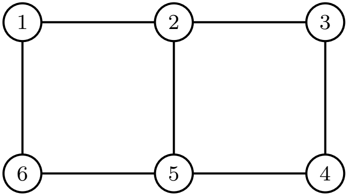
The parity matrix $P_{8\times 7}$ and connectivity graph G.

The 7th Row Encodes Bit Flip

$$\begin{aligned}\mathcal{P} \quad = \quad & \{(1, \textcolor{red}{1} \oplus x_1 \oplus x_4 \oplus x_5), (2, x_2 \oplus x_3 \oplus x_5 \oplus x_6), (4, \textcolor{red}{1} \oplus x_4 \oplus x_5 \oplus x_6), (4, \textcolor{red}{1} \oplus x_1 \oplus x_2 \oplus x_6), \\ & (6, \textcolor{red}{1} \oplus x_1 \oplus x_2 \oplus x_3), (7, \textcolor{red}{1} \oplus x_1 \oplus x_2 \oplus x_4 \oplus x_6), (\textcolor{red}{1}, x_2 \oplus x_4 \oplus x_5)\}\end{aligned}$$

$$P = \begin{array}{c} \begin{array}{ccccccc} \textcolor{green}{p_1} & \textcolor{green}{p_2} & \textcolor{green}{p_3} & \textcolor{green}{p_4} & \textcolor{green}{p_5} & \textcolor{green}{p_6} & \textcolor{green}{p_7} \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ \textcolor{red}{1} & \textcolor{red}{0} & \textcolor{red}{1} & \textcolor{red}{1} & \textcolor{red}{1} & \textcolor{red}{1} & \textcolor{red}{0} \\ 1 & 2 & 4 & 4 & 6 & 7 & 1 \end{bmatrix} \end{array}$$

$G =$



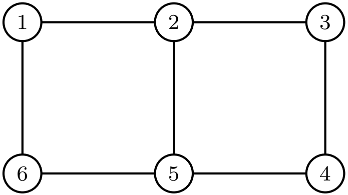
The parity matrix $P_{8\times 7}$ and connectivity graph G.

The Last Row Stores Coefficients

$$\mathcal{P} = \{(1, 1 \oplus x_1 \oplus x_4 \oplus x_5), (2, x_2 \oplus x_3 \oplus x_5 \oplus x_6), (4, 1 \oplus x_4 \oplus x_5 \oplus x_6), (4, 1 \oplus x_1 \oplus x_2 \oplus x_6), (6, 1 \oplus x_1 \oplus x_2 \oplus x_3), (7, 1 \oplus x_1 \oplus x_2 \oplus x_4 \oplus x_6), (1, x_2 \oplus x_4 \oplus x_5)\}$$

$$P = \begin{array}{c} \begin{array}{ccccccc} \underline{p_1} & \underline{p_2} & \underline{p_3} & \underline{p_4} & \underline{p_5} & \underline{p_6} & \underline{p_7} \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 2 & 4 & 4 & 6 & 7 & 1 \end{bmatrix} \end{array}$$

$G =$



The parity matrix $P_{8\times 7}$ and connectivity graph G.

PHASE-NW-SYNTH Algorithm Snapshot

- Ignore the last two rows of P, let $B = \{p'_1, p'_2, p'_3, p'_4, p'_5, p'_6, p'_7\}$, $\mathcal{K}$ be an empty stack, and $I = [6]$.
- Cycle through the set of n-bit strings and apply corresponding $CNOT$ gates at each iteration.
- Whenever a column has a single 1, it implies that the corresponding parity has been realized.

Example After the 4th iteration, we have

$$
B^{(4)} = \begin{array}{c}
\begin{array}{ccccccc} p'_1 & p'_2 & p'_3 & p'_4 & p'_5 & p'_6 & p'_7 \end{array} \\
\begin{bmatrix}
1 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 1
\end{bmatrix}
\end{array}
$$

PHASE-NW-SYNTH Algorithm Snapshot

- Whenever a column has a single 1, it implies that the corresponding parity has been realized.
- Remove these columns from the remaining parities.
- Place the gate X if parity realized on circuit is $1 \oplus f$ for some $(c, f) \in \mathcal{P}$. We can also place a gate in $\{\mathsf{T}, \mathsf{T}^\dagger, \mathsf{S}, \mathsf{S}^\dagger, \mathbb{Z}, \mathsf{Y}\}$ corresponding to the value of the coefficient c.

Example The partial circuit obtained after applying a sequence of gates from iteration 4.

$$B^{(4)} = \begin{array}{c} \begin{array}{ccccccc} p'_1 & p'_2 & p'_3 & p'_4 & p'_5 & p'_6 & p'_7 \end{array} \\ \begin{bmatrix} 1 & 0 & \color{red}{0} & 1 & 1 & 1 & 0 \\ 0 & 1 & \color{red}{0} & 1 & 1 & 1 & 1 \\ 0 & 1 & \color{red}{0} & 0 & 1 & 0 & 0 \\ 1 & 0 & \color{red}{1} & 0 & 0 & 1 & 1 \\ 0 & 1 & \color{red}{0} & 0 & 0 & 1 & 0 \\ 1 & 0 & \color{red}{0} & 1 & 0 & 1 & 1 \end{bmatrix} \end{array}$$