COE 530
Quantum Computers and Architecture

Lecture 8

Quantum Circuit Synthesis – Unitary Decomposition

References:

Ch 6 Quantum Computing Systems

Generic Quantum Compilation Flow

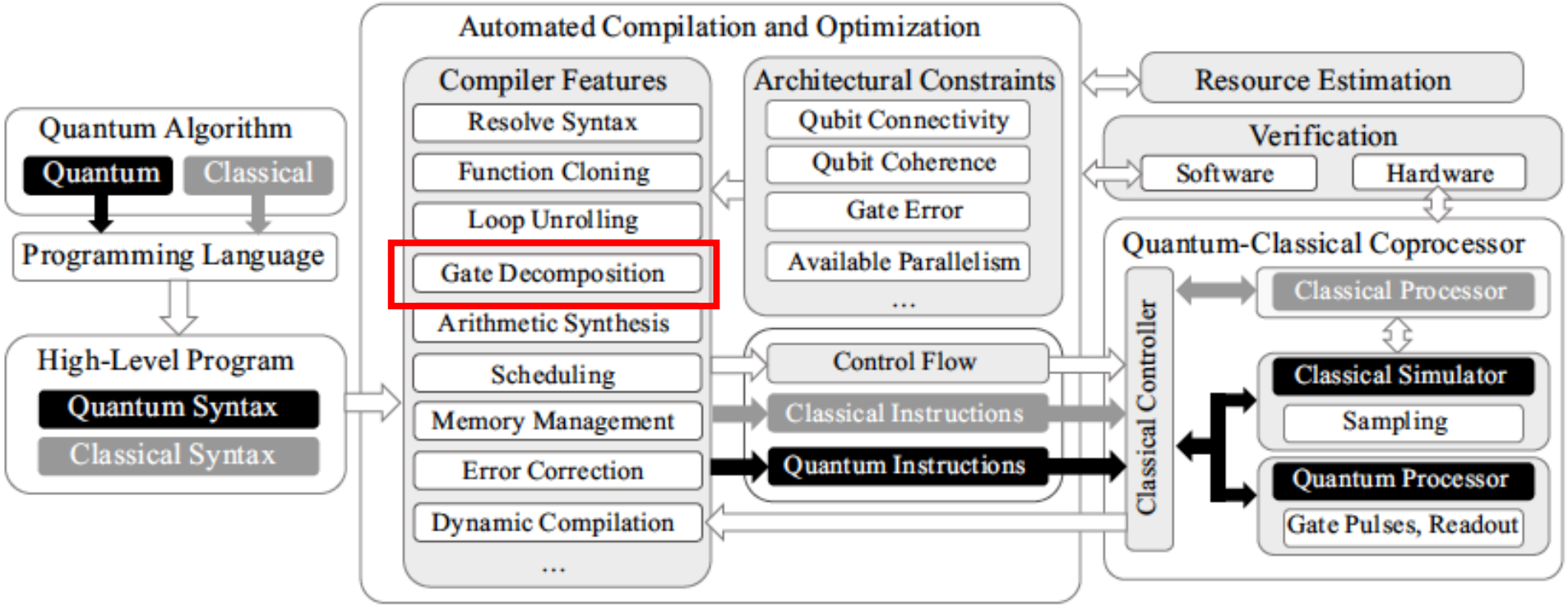


Figure 4.2: A detailed quantum compilation flow outlining the transformations and optimizations involved in a generic compiler.

Quantum Computer Systems: Research for Noisy Intermediate-Scale Quantum Computers

Quantum algorithm implementation pipeline.

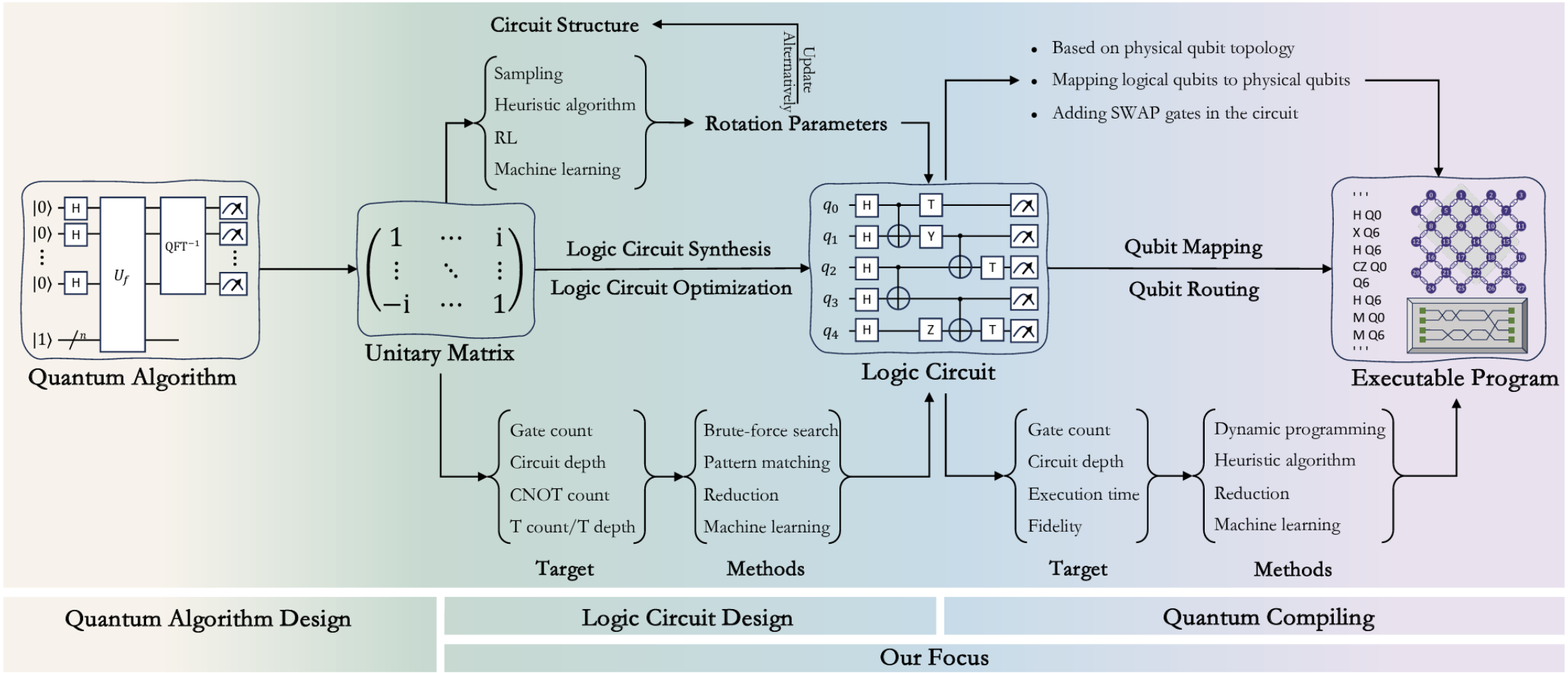


Fig. 1. Quantum algorithm implementation pipeline. A quantum algorithm can be written in the form of multiple unitary transformations. logic circuit synthesis and optimization methods are then applied to obtain logic circuits. We utilize qubit mapping and routing methods to build an executable program during the quantum compiling stage. In this paper, we mainly focus on logic circuit design and quantum compiling.

https://arxiv.org/pdf/2407.00736

Circuit Representations: Overview

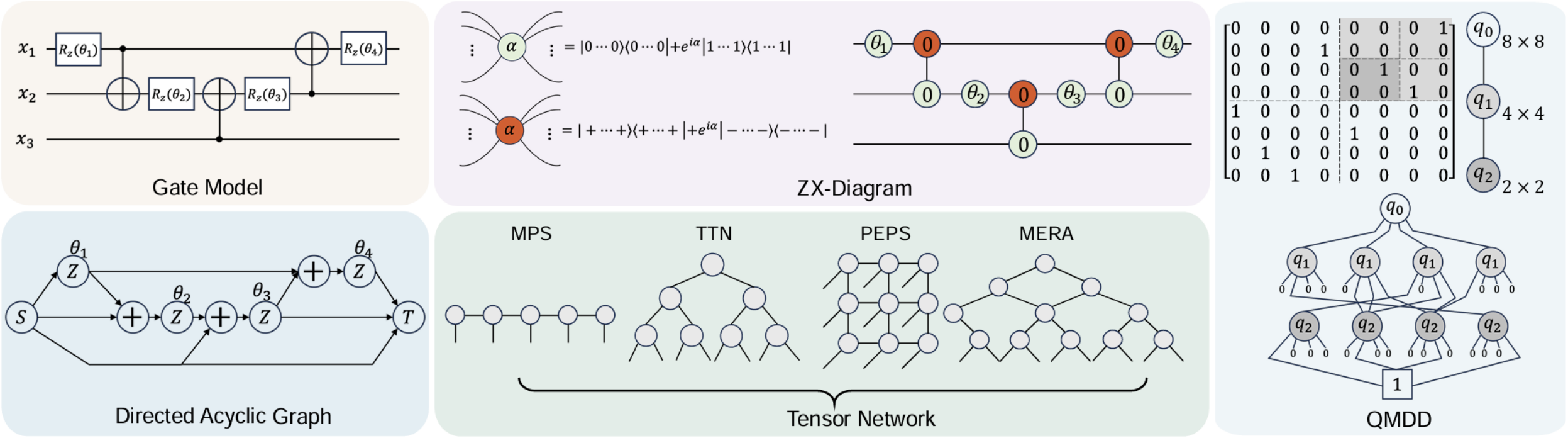


Fig. 2. Quantum Circuit Representation. Gate Model: a popular representation method; DAG: example DAG transformed from the gate model example, "start" and "end" nodes are added to complete the graph; QMDD: QMDD representation of a three-qubit quantum operation; we present the transformation matrix, the layerwise structure and the size of the corresponding matrix of each node; Tensor Network: a quantum state $|\psi\rangle$ can be represented by multiple common types of tensor network; ZX-Diagram: the green and red spider in ZX-diagram as well as an example ZX-diagram transformed from gate model example.

https://arxiv.org/pdf/2407.00736

Circuit Representations: Overview

- Quantum circuits can be represented at multiple levels of abstraction

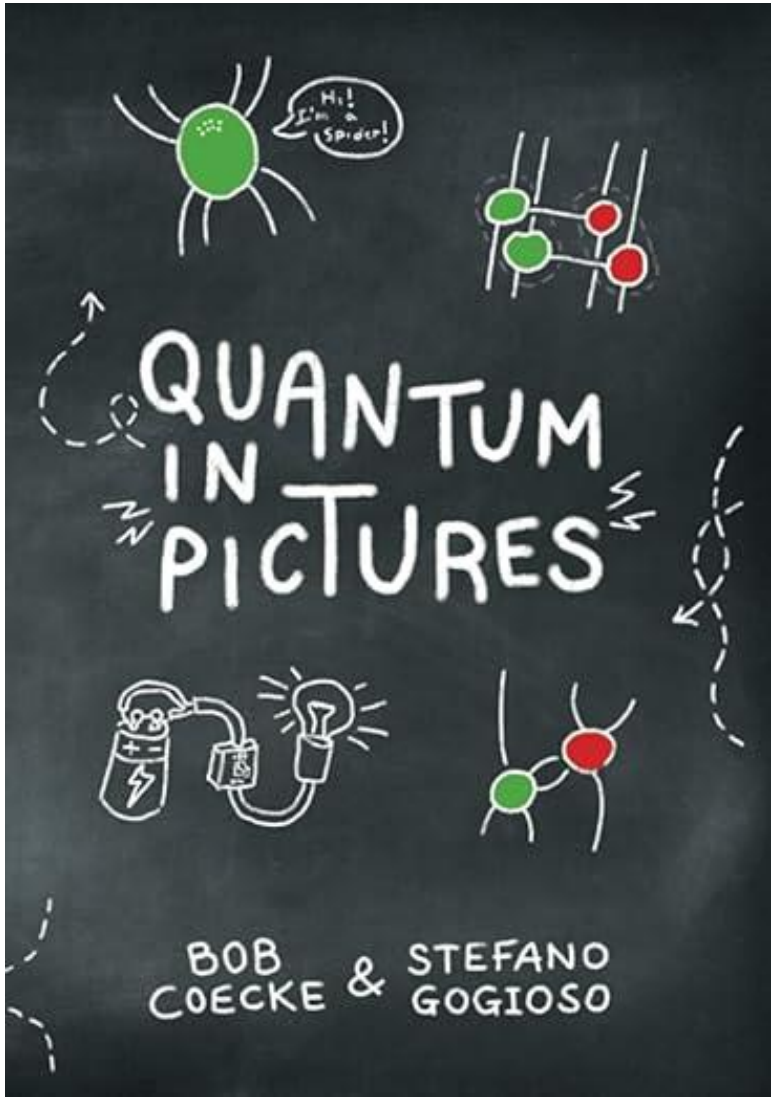
- Unitary Matrix
  - $U \in SU(2^n)$: the mathematical specification of the desired transformation
  - Exponential in size ($2^n \times 2^n$) — impractical for large n

- Gate-Level Circuit (Diagram)
  - Sequence of gates from a universal gate set, e.g., $\{H, T, CNOT\}$
  - Horizontal wires = qubits, boxes/symbols = gates, read left to right

- DAG (Directed Acyclic Graph)
  - Nodes = gates, edges = data dependencies between gates on shared qubits
  - Enables scheduling, parallelism analysis, and optimization passes

- ZX-Diagrams
  - ZX-diagrams are a method for representing quantum circuits based on ZX calculus
  - A special form of tensor networks that can represent any linear mapping between qubits
  - ZX diagrams include two types of spiders: green Z spiders and red X spiders

To learn more about ZX diagrams



Quantum Logic Curcuit Synthesis

- Quantum logic circuit synthesis is the process of translating the quantum algorithm's unitary matrix into low-level gate (i.e. $X, CNOT$) gates that can be physically executed by a quantum processor

- Curcuit synthesis depends on :
  - Choice of universal instruction set S
  - Single-qubit and multi-qubit
  - Exact and approximate synthesis

- Key synthesis approaches:
  - Exact decomposition (CSD, QSD)
  - Approximate synthesis (Solovay-Kitaev, Clifford+T, Ross-Selinger)
  - Quantum Architecture Search (ML/RL-driven circuit design for VQAs)

Universal Gate

- Recall openQASM3.0, the built-in single-qubit gate is

$$U(\theta, \varphi, \lambda) = \begin{bmatrix} \cos(\frac{\theta}{2}) & -e^{i\lambda}\sin(\frac{\theta}{2}) \\ e^{i\phi}\sin(\frac{\theta}{2}) & e^{i(\phi+\lambda)}\sin(\frac{\theta}{2}) \end{bmatrix}$$

  - θ: Controls the rotation around the Y-axis (latitude)
  - ϕ: Controls a rotation around the Z-axis after θ
  - λ: Controls an initial rotation around the Z-axis before θ
- Combined with CX(CNOT), these two gates form a universal set, i.e., any quantum circuit can be built from $U + CX$
- But how is U implemented at low-level?

Standard Gate	OpenQASM Equivalent
Identity I	$U(0, 0, 0)$
Pauli-X	$U(\pi, 0, \pi)$
Hadamard H	$U(\pi/2, 0, \pi)$
Pauli-Z	$U(0, 0, \pi)$
Phase S	$U(0, 0, \pi\mathrm{i}/2)$

Universal Instruction (Gate) Set

- **Definition:** *A quantum instruction set S is called computationally universal if and only if gates from S allow the realization of an arbitrary quantum circuit C.*

- A set of gates is universal if any unitary operation can be approximated to any desired accuracy using only gates from that set
- The "Gold Standard" (Clifford + T):The most common universal set used in fault-tolerant research consists of:
  - H: Creates superposition
  - S: A π rotation around the Z-axis
  - $CNOT$: Entangles two qubits
  - T-gate: A $\frac{\pi}{4}$ rotation around the Z-axis (Why the T-gate?)
- Other Examples:
  - Discrete Set: $\{H, T, CNOT\}$ — Best for error-corrected systems.
  - Continuous Set: $\{RX(\theta), RY(\theta), CNOT\}\$$ — Often used in NISQ (Noisy Intermediate-Scale Quantum) devices where we can tune the rotation angle θ precisely.

Example – IBM Universal Gate Set

- AS IBM has moved to the newer **Heron** and **Nighthawk** processors, the "Universal Gate Set" at the hardware level has fundamentally shifted
- The new universal gate set is $\{RZ, SX, X, CZ\}$
  - $RZ(\lambda)$ (Virtual-Z)
    - "Zero-Cost" Gate: This is a phase shift implemented by simply changing the phase of the microwave drive in the classical controller
    - Duration: 0 ns. Error: 0%
  - SX $\left(\sqrt{X}\right)$
    - A $\frac{\pi}{2}$ rotation around the X-axis
    - This is the physical "building block" used to create Hadamards and other rotations
  - X
    - A full π
    - This is often just two $SX$$SX$ pulses back-to-back
  - CZ (Controlled-Z):
    - Unlike the old CX (which used cross-resonance), CZ is implemented using tunable couplers that bring two qubits into resonance momentarily. It is faster and higher-fidelity
    - $CNOT_{0,1} = (I \otimes H)CZ_{0,1}(I \otimes H)$

Unitary Decomposition

- Unitary decomposition is to take a large quantum operation U (an $N \times N$ unitary, $N = 2^{n}$) and rewrite it as a product of smaller gates
- Goal: $U \approx G_{m} G_{m-1} \cdots G_{2} G_{1}$ where each G_{i} is a 1-qubit or 2-qubit gate from your allowed set

Why Decompose U?

- A general n-qubit unitary is $2^{n} \times 2^{n}$, grows exponentially
- Hardware can only execute short sequences of elementary 1- and 2-qubit gates
- Decomposition bridges the gap between abstract unitaries and physical hardware

Exact Unitary Decomposition Methods

1. Two-Level / Givens Rotation Decomposition

2. Cosine Sine Decomposition / Quantum Shannon Decomposition

3. KAK / Cartan Decomposition

Approach 1 – Two-Level / Givens Rotation Decomposition

Core Idea

- Reduce U to the identity by zeroing matrix entries step-by-step
- Each step uses a "two-level unitary" that acts on only a 2D subspace
- Translate resulting two-level unitaries into controlled rotations + $CNOT$s

Example – Decomposing a 2-Qubit (4×4) Unitary

- Given a 4×4 unitary U, apply a Givens rotation G_{12} to zero entry $U(1,2)$
- Then G_{13} to zero $U(1,3)$, then $G_{14}, G_{23}, G_{24}, G_{34}$
- Result: $G_{34}\, G_{24}\, G_{23}\, G_{14}\, G_{13}\, G_{12} \cdot U = I \rightarrow U = G_{12}^{\dagger} G_{13}^{\dagger} G_{14}^{\dagger} G_{23}^{\dagger} G_{24}^{\dagger} G_{34}^{\dagger}$
- Each $G_{ij}^{\dagger}$ maps to a controlled rotation + CNOT on the 2-qubit register

Numerical Example – Givens Rotation Decomposition

Goal: Zero out $U(2,1)$ using a Givens rotation

Suppose our 2×2 unitary is: $U = \begin{bmatrix} 0.6 & -0.8 \\ 0.8 & 0.6 \end{bmatrix}$ (a simple rotation by $\theta \approx 53.13°$)

Step 1 – Construct the Givens rotation to eliminate $U(2,1) = 0.8$

- Compute: $r = \sqrt{0.6^2 + 0.8^2} = \sqrt{0.36 + 0.64} = 1$
- Givens angle: $\cos\varphi = 0.6/1 = 0.6, \quad \sin\varphi = 0.8/1 = 0.8$
- Build: $G = \begin{bmatrix} 0.6 & 0.8 \\ -0.8 & 0.6 \end{bmatrix}$ (rotation by $-\varphi$)

Step 2 – Multiply: $G \cdot U = I$

- Row 1: (0.6)(0.6)+(0.8)(0.8) = 1, (0.6)(−0.8)+(0.8)(0.6) = 0
- Row 2: (−0.8)(0.6)+(0.6)(0.8) = 0, (−0.8)(−0.8)+(0.6)(0.6) = 1
- So $G \cdot U = I$, therefore $U = G^{\dagger} = [\,0.6, -0.8\,;\ 0.8, 0.6\,]$

Key takeaway: For larger $N \times N$, repeat this process $\sim N(N-1)/2$ times to reduce $U \rightarrow I$

Approach 2 – Cosine Sine Decomposition / Quantum Shannon Decomposition

Core idea

- Breaks U into: $diag(U_1, U_2) \cdot \begin{bmatrix} C & -S \\ S & C \end{bmatrix} \cdot diag(V_1, V_2)$
- Middle block → multiplexed rotations; block-diag → recurse on (n−1) qubits
- $C = diag(cos\theta_1, \dots, cos\theta_n), S = diag(sin\theta_1, \dots, sin\theta_n)$, and $C^2 + S^2 = I$

Example – 2-Qubit (4×4) Unitary via CSD

- Partition the 4×4 matrix into four 2×2 blocks (top-left, top-right, etc.)
- CSD gives: $U = diag(U_1, U_2) \cdot R_x(\theta) \cdot diag(V_1, V_2)$ with a single angle θ
- $diag(U_1, U_2)$ → two independent 1-qubit gates controlled on qubit 0
- $R_x(\theta)$ → a single $CNOT + Ry(\theta) + CNOT$ (uniformly-controlled Ry)
- For 3+ qubits, each 2×2 block is itself a unitary — recurse until 1-qubit

Numerical Example – CSD (2-Qubit = 4×4 Case)

Setup: Decompose a 4×4 unitary (2 qubits) via CSD

Let $U = Hadamard \otimes I = 1/\sqrt{2}\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$

Step 1 – Partition into 2×2 blocks

- Top-left $A = (1/\sqrt{2})\, I$, Top-right $B = (1/\sqrt{2})\, I$
- Bottom-left $C = (1/\sqrt{2})\, I$, Bottom-right $D = -(1/\sqrt{2})\, I$

Needs verification!

Step 2 – Apply CSD: $\boldsymbol{U = diag(U_1, U_2) \cdot \begin{bmatrix} C & -S \\ S & C \end{bmatrix} \cdot diag(V_1, V_2)}$

- SVD of top-left gives:
- , $U_2 = V_2 = I$

Result: Middle block is a single $Ry(\pi/4)$ controlled on qubit 0

Approach 3 – KAK / Cartan Decomposition (2-Qubit)

Core idea

- $U = (A_1 \otimes A_2) \cdot \exp[i(\alpha(X \otimes X) + \beta(Y \otimes Y) + \gamma(Z \otimes Z))] \cdot (B_1 \otimes B_2)$
- Local gates A, B are single-qubit; middle "entangling core" uses ≤ 3 CNOTs
- The values (α, β, γ) are called Cartan coordinates

Example – Decomposing CNOT via KAK

- The CNOT gate is itself a 2-qubit unitary
- Apply KAK decomposition:
  - KAK parameters: $\alpha = \pi/4,\ \beta = 0,\ \gamma = 0$
  - Local gates: $A_1 = Ry(\pi/2),\ A_2 = I,\ B_1 = Rz(-\pi/2),\ B_2 = Rz(-\pi/2)$
  - Since β = γ = 0, the entangling core needs only 1 CNOT (not the max of 3)
- Key insight: KAK reveals how "entangling" a gate is via its (α, β, γ) parameters
- Kak decomposition determines the absolute minimum number of CNOTs needed to build a specific gate

Exact Unitary Decomposition Methods – Summary

Feature	**Givens Rotation**	**CSD / QSD**	**KAK / Cartan**
Strategy	Element-wise elimination (Bottom-Up)	Recursive subspace splitting (Top-Down)	Geometric Lie algebra mapping
Best For	Proving universality 2×2 matrices	Arbitrary n-qubit unitaries	**Optimal** 2-qubit synthesis
CNOT Count	$O(n^3 4^n)$ (High)	$\approx \frac{1}{2} 4^n$$ (Near-Optimal)	**3 CNOTs** (Minimum)
Complexity	Simple, iterative	Highly recursive, algebraic	Highly geometric/abstract
Exactness	Exact	Exact	Exact

Why Do We Need the Solovay-Kitaev Theorem?

The Problem: Finite Gates vs. Continuous Unitaries

- A universal gate set (e.g., {H, T}) can produce only countably many unitaries
- But $SU(2)$ is continuous — uncountably many single-qubit gates exist
- So we can never implement most gates exactly — only approximate them

The Key Question

- How many gates from {H, T} do we need to approximate any U to accuracy ε?

Naïve Approach: Brute-Force Search

- Enumerate all gate sequences of length ℓ, pick the closest to U
- Requires sequence length $\ell = O(1/\varepsilon)$ to achieve accuracy ε — impractical

What Solovay-Kitaev Gives Us

- A dramatically better scaling: only $O(\log^c\left(\frac{1}{\varepsilon}\right))$ gates, where c ≈ 3.97
- Exponentially fewer gates than brute force — makes approximation practical

Solovay-Kitaev Theorem & Algorithm

Theorem (Solovay-Kitaev)

- Let G be a finite gate set that generates a dense subgroup of $SU(2)$
- Then for any $U \in SU(2)$ and $\varepsilon > 0$, there exists a sequence of $O(\log^c(1/\varepsilon))$ gates from G that approximates U to within ε, where c ≈ 3.97

The Algorithm (Recursive)

- Base case: look up a "rough" approximation U_0 from a precomputed table (accuracy ε_0)
- Recursion: to improve from ε_n to $\varepsilon_{n+1} \approx \varepsilon_n^{\frac{3}{2}}$:
  - 1. Compute the "error" $U_{err} = U \cdot U_n^{\dagger}$
  - 2. Factor U_{err} as a group commutator: $U_{err} \approx VWV^{\dagger}W^{\dagger}$
  - 3. Recursively approximate V and W
  - 4. $U_{n+1} = \tilde{V}\tilde{W}\tilde{V}^{\dagger}\tilde{W}^{\dagger} \cdot U_n$ (improved approximation)

Key insight: Commutator $VWV^{\dagger}W^{\dagger}$ has error ≈ ε², so each recursion squares the accuracy

Next

- Oracle synthesis
- AI/ML approached for circuit synthesis
- Quantum Architecture Search