Compilation, Depth, and Width

Variational Quantum Algorithms · Module 2 — Parameterized Circuits

Node 2.4

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Parameterized Circuits **Node:** 2.4

The mathematical ansatz $U(\boldsymbol{\theta})$ and the physical circuit that runs on a device are not the same thing. Between them sits a **compiler** — a piece of software that translates the abstract gate sequence into one the hardware can actually execute, respecting native gate set, qubit connectivity, scheduling constraints, and calibration.

This node is the bridge from "ansatz on paper" to "circuit on metal." The relevant concepts are **width** (how many qubits), **depth** (how many sequential gate layers), and **gate count** (how many physical operations). On NISQ devices, all three are budgets, and compiling well means staying inside all three at once.

Width

Width is the number of qubits the circuit acts on. On a real device, width is bounded by the number of available, well-calibrated qubits.

Three subtleties:

- **Logical vs physical qubits**: every algorithm-level qubit consumes one device qubit — no error-correction overhead in the NISQ setting. (Fault-tolerant compilation is a different story.)
- **Connectivity**: not all qubit pairs can interact directly. A "100-qubit device" might only support entangling gates between specific neighbors. If your ansatz wants a CNOT between two qubits that aren't connected, the compiler must insert SWAPs to bring them adjacent — and each SWAP is itself a CNOT-heavy operation.
- **Calibration heterogeneity**: not all qubits are equally good. A 27-qubit device might only have 12 qubits with low enough error rates to be worth using on a given day.

Width is the **easiest budget to violate** — your problem either fits or it doesn't — and the **hardest to compress** without physically larger hardware.

Depth

Depth is the length of the longest path through the circuit (the number of gate layers that must execute sequentially). On a real device, depth is bounded by the **coherence time** — the qubits remain in a usable quantum state only for a finite duration before decoherence destroys the information.

A typical NISQ device has:

- Single-qubit gate time: ~20 ns
- Two-qubit gate time: ~200-500 ns
- T1/T2 coherence: 50-200 µs

This gives a depth budget of **a few hundred two-qubit gates** before the state is irrecoverably noisy. For a 100-CNOT circuit, you have already consumed most of the coherence window.

Depth is the most-often-violated budget for chemistry-style ansätze, which is why UCCSD is hard to run on real hardware and ADAPT-VQE (and other depth-saving variants) exist.

Critical distinction: depth vs gate count. A circuit with 1000 single-qubit gates spread across 10 qubits has gate count 1000 but depth only ~100, because gates on different qubits run in parallel. Depth is the time-axis quantity; gate count is the resource quantity. Both matter, for different reasons.

Gate Count

Gate count is the total number of operations. It governs **noise accumulation** — each gate has some error probability p, so the circuit fidelity scales roughly as $(1-p)^{\text{gate count}} \approx e^{-p \cdot \text{gate count}}$.

For $p = 10^{-3}$ (typical NISQ two-qubit error) and a 1000-CNOT circuit, the fidelity is $e^{-1} \approx 0.37$ — meaning roughly 60% of your runs are corrupted by at least one error. For a 100-CNOT circuit, fidelity is $e^{-0.1} \approx 0.90$ — much more workable.

Gate count is what makes "shallow" ansätze attractive: not because they finish faster (a single circuit is fast either way), but because they accumulate less error per shot.

What the Compiler Does

A compiler takes an abstract $U(\boldsymbol{\theta})$ and produces a runnable circuit. Its responsibilities, in roughly the order they happen:

1. Decompose abstract gates into native gates. $R_Y(\theta)$ might not be native; only R_X, R_Z, and CZ might be. The compiler rewrites $R_Y(\theta)$ using the available basis: $R_Y(\theta) = R_Z(-\pi/2)R_X(\theta)R_Z(\pi/2)$.

2. Route logical qubits to physical qubits. The compiler picks an assignment of algorithm qubits to device qubits that minimizes SWAP overhead given the device connectivity graph.

3. Insert SWAPs where needed. For two-qubit gates between non-adjacent qubits, SWAPs move the qubits adjacent. Each SWAP = 3 CNOTs, so this is expensive.

4. Schedule. Decide which gates run in parallel (on different qubits) and which run sequentially. Minimizes depth.

5. Optimize. Combine adjacent rotations, cancel redundant CNOTs, exploit gate identities. Modern compilers use heuristic and SAT-based passes.

6. Calibrate-aware mapping. Pick the best available qubits given today's noise profile.

The compiled circuit can be **2-10× larger** than the abstract one for non-trivial connectivity, especially for chemistry-style ansätze that want all-to-all entanglement on a device with linear or grid connectivity.

The VQA Compilation Subtlety

In a normal quantum algorithm, you compile once, run many times. In a VQA, the *parameters* change between runs but the *structure* doesn't — so a smart VQA compiler compiles the **structure** once and only re-evaluates the parameter angles each iteration.

Modern toolchains (Qiskit, Pennylane, Cirq) all support this "parameterized compilation" pattern: build a circuit template with symbolic parameters, compile to native gates symbolically, then bind concrete parameter

values per call. Without this, every optimizer step would pay full compilation cost — potentially seconds — and the inner loop would be unworkable.

This is one of those infrastructure details that doesn't show up in any paper but determines whether a 1000-iteration VQA finishes in an hour or a week.

Why This Matters For Optimizer Choice

Some optimizers ask for many evaluations per iteration (population methods, finite-difference gradient estimation): these are bottlenecked by per-evaluation cost. Others ask for few high-quality evaluations: these are bottlenecked by per-evaluation accuracy.

If your compiled circuit is large enough that every shot is expensive (e.g., it takes 100 ms of wall clock and consumes a chunk of your queue allocation), then **the right optimizer is one that uses few evaluations** — Bayesian optimization, surrogate methods, gradient methods with parameter-shift rather than finite differences.

If your circuit is small and cheap, you can afford a population method or a high-shot-budget gradient method.

The compiled circuit is the constraint that makes this tradeoff concrete. **You can't reason about VQA optimization in isolation from compilation cost** — it sets the per-iteration budget that all of the algorithmic choices have to fit inside.

Structural Role

This is the **operational reality check** node of Module 2. It connects the abstract ansatz definitions of 2.1-2.3 to the physical hardware that nodes 2.7-2.8 will describe in detail. Without this node, the rest of Module 2 reads as pure mathematics; with it, the design tradeoffs for the rest of the course have material weight.

It also feeds Module 9 (hardware noise models): noise rates and gate counts compose into the effective objective function noise profile that the optimizer faces.

Relations to Other Concepts

- **Quantum compilation** — the general subfield. Major tools: Qiskit transpiler, Pennylane transforms, Cirq optimizers, t|ket⟩.
- **Topological compilation** — fault-tolerant compilation for surface codes; not relevant in NISQ.
- **Gate teleportation / measurement-based computation** — alternative compilation models; not yet widespread.
- **Pulse-level compilation** — bypassing gate-level abstractions to compose at the analog pulse level; under active development.

Worked Example — UCCSD on a Linear Topology

Consider a 4-qubit H_2 UCCSD circuit. Abstract form: 1 Pauli rotation $e^{i\theta P}$ where P is an 8-Pauli-string sum acting on all 4 qubits.

After abstract decomposition: - 8 Pauli rotations (one per term in P, Trotterized to first order), - Each Pauli rotation = (basis change) + (CNOT staircase, length 3) + (R_Z) + (inverse staircase) + (inverse basis change), - ~7 CNOTs per term × 8 terms = ~**56 CNOTs**.

On a device with **all-to-all connectivity** (e.g., trapped ion), the compiled circuit is roughly that size. On a device with **linear connectivity** (qubits in a chain, 1-2-3-4), CNOTs between non-adjacent qubits need SWAPs: - A CNOT between qubits 1 and 4 costs 3 + 1 + 3 = 7 native CNOTs (SWAP across, do the gate, SWAP back). - If the staircase requires multiple non-adjacent CNOTs, the SWAP overhead compounds. - Compiled CNOT count: ~**120-180**, depending on routing.

So the same abstract UCCSD ansatz costs 2-3× more on a linear device than on a fully-connected one. This is the operational reason that trapped-ion devices are sometimes more competitive for chemistry than larger but less-connected superconducting devices, even at lower qubit count.

Visualization

Pending. Two-panel diagram. Left: an abstract circuit (4 qubits, a few gates, clean). Right: the same circuit after compilation onto a linear-topology device — many SWAPs inserted, longer overall depth. Caption: "Connectivity is a tax."

Exercises

1. Estimate the compiled depth and CNOT count for a 6-qubit HEA with 3 layers on a square 2×3 grid topology, assuming each layer's entangling pattern is a complete bipartite graph.
2. For a CNOT error rate of $p = 5 \times 10^{-3}$, what is the maximum CNOT count for which the circuit fidelity remains above 0.5?
3. (VQA) The "compile once, bind many" pattern is critical for VQA performance. Sketch a minimal API for parameterized compilation that supports gradient evaluation via the parameter-shift rule without recompilation.

Research Extensions

- **Approximate compilation** — accept some compilation error in exchange for shorter circuits; relevant for ansatz-level approximations.
- **Hardware-aware ansatz synthesis** — automatically construct an ansatz given a target Hamiltonian and a device connectivity graph.
- **Learned compilation** — RL-based compilers that improve on heuristic SWAP insertion.
- **Pulse-level VQAs** — bypass gate-level decomposition by directly optimizing pulse parameters.

Cross-links to Other Courses

- **Compilers (CS)** — instruction selection, register allocation, scheduling — the same algorithmic problems with quantum-specific cost models.
- **Module 9 (Hardware Noise Models)** — gate count maps to noise budget.
- **Quantum Hardware** — coherence times, native gate sets, connectivity graphs.
- **Operations Research** — SWAP minimization is a hard combinatorial problem.

Variational Quantum Algorithms · Module 2 — Parameterized Circuits · Node 2.4
Concepts: parameterized-circuits, computational-complexity, graph-theory
Prerequisites: 2.1
Enables: 2.7

hbar.university — own.your.cognition