Compiling Quantum Programs

Physics Phryday Presentation

Li-Heng Henry Chang

Computer Science and Physics Department,
Bard College

May 5, 2023

TABLE OF CONTENT

Why Quantum Compilation?

- The need for different levels of abstraction/control to explore quantum architecture, which is under quick development.
- Portable code that can be used to benchmark hardware platforms.
- Faster run times to avoid decoherence that destroys computational results.
- Happy programmer, developer and students means more problems solved!

Quantum Computation

What is a qubit?

The smallest unit of information to keep track of a quantum state is a "quantum bit" or "qubit".

The special states $|0\rangle$ and $|1\rangle$ are called the computational basis states that form an orthonormal basis

$$|0\rangle \to \begin{bmatrix}1\\0\end{bmatrix}, |1\rangle \to \begin{bmatrix}0\\1\end{bmatrix}.$$

Mathematically, a single-qubit state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ can be described by two complex numbers such that they are normalized $|\alpha|^2 + |\beta|^2 = 1$.

$$|\psi\rangle \to \alpha\begin{bmatrix}1\\0\end{bmatrix} + \beta\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}\alpha\\\beta\end{bmatrix}$$

The amplitudes squared $|\alpha|^2$ and $|\beta|^2$ are the probabilities of obtaining the $|0\rangle$ and $|1\rangle$ states.

QUANTUM COMPUTATION

WHAT IS A QUBIT?

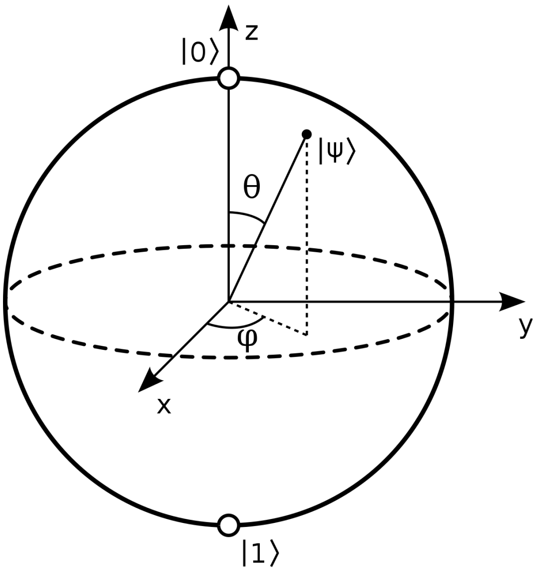

Figure. The Bloch Sphere Visualization of a single-qubit state. Since $|\alpha|^2 + |\beta|^2 = 1$, we can rewrite $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ as $|\psi\rangle = e^{i\gamma}\left(\cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle\right)$. This parameterizes $|\psi\rangle$ in terms of two angles θ and ϕ that define a point on the unit R^3 sphere. This is called a *Bloch Sphere* that provides a useful way of visualizing a single-qubit state.

How do we describe the evolution U of a state over time $|\psi'\rangle = U|\psi\rangle$? The normalization condition says that the transformation U has to be *unitary* $U^\dagger U = I$. Let's see an example here.

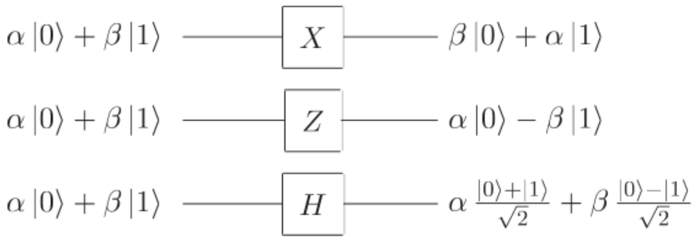

Figure. Circuit diagrams of single-qubit gates.

In matrix formulation,

$$X\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}\beta\\ \alpha\end{bmatrix} \to X = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$$

Other examples

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}, Z = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}, T = \begin{bmatrix}1 & 0\\ 0 & \exp\left(\frac{i\pi}{4}\right)\end{bmatrix}.$$

Two single-qubit states can be combined as

$$|ba\rangle = |b\rangle \otimes |a\rangle = \begin{bmatrix} b_0 \times \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \\ b_1 \times \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} b_0a_0 \\ b_0a_1 \\ b_1a_0 \\ b_1a_1 \end{bmatrix}$$

If we apply this to the single-qubit computational basis state, we can get the computational basis states for two-qubit systems.

$$|0\rangle \otimes |0\rangle = |00\rangle \rightarrow \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, |0\rangle \otimes |1\rangle = |01\rangle \rightarrow \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, |1\rangle \otimes |0\rangle = |10\rangle \rightarrow \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, |1\rangle \otimes |1\rangle = |11\rangle \rightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

A useful two-qubit operation is

$$\text{CNOT}\begin{bmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{bmatrix} = \begin{bmatrix} a_{00} \\ a_{01} \\ a_{11} \\ a_{10} \end{bmatrix} \begin{matrix} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{matrix} \rightarrow \text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

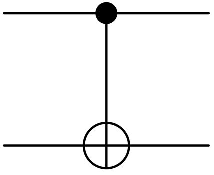

Figure. Circuit diagram and matrix representation of the CNOT gate.

The state space size is 2^n for a n-qubit state!

An algorithm can be represented as a matrix

$$U = \begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix},$$

or a circuit diagram.

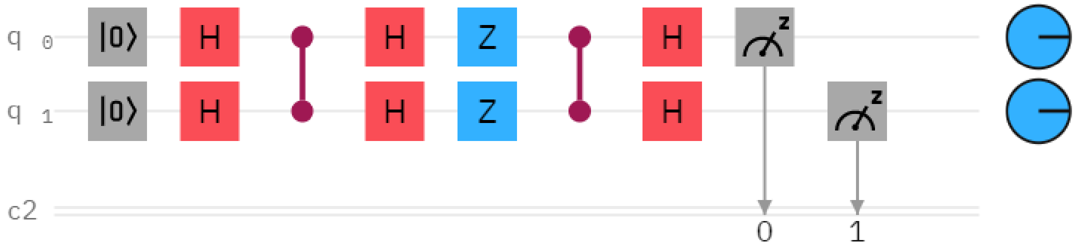

Figure. The circuit diagram of Grover's algorithm.

QUANTUM COMPILATION

A quantum algorithm is a unitary transform U on a quantum state $|\psi\rangle$ that produces a computed state $|\psi'\rangle = U|\psi\rangle$. However, we are only given a limited discrete set of gates G by physical hardware. How can we realize U using G, with a small error ϵ?

For example U is CCNOT operation, and $G = \{CNOT, H, T\}$?

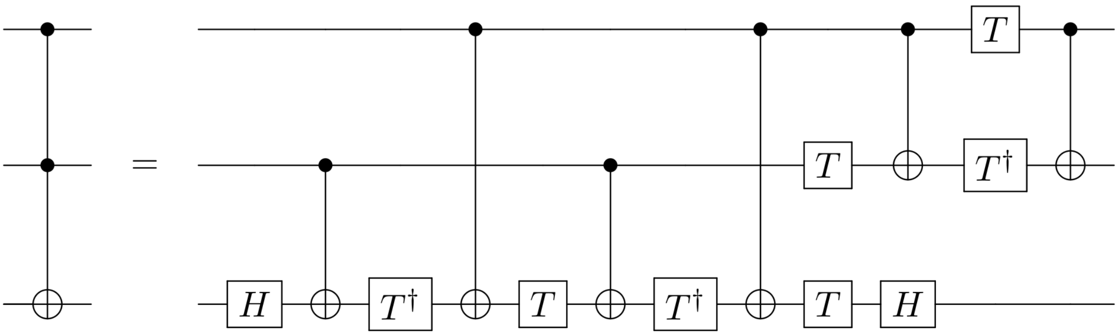

Figure. The compilation of CCNOT gate into CNOT, H and T gates.

Can we do this more generally? Can we find G that can approximate any algorithm U?

Universal Quantum Computation

A set of gates is **universal** for quantum computation if any unitary matrix can be approximated to arbitrary accuracy by a quantum circuit built with only those gates.

We refer to the fact that there exists a universal gate set as the **universality of quantum computation**, which follows from the three statements below:

- An arbitrary unitary operator U of d dimensions may be expressed *exactly* as a product of 2-level unitary operators. An n-qubit system may be written as a product of $O(4^n)$ two-level unitary matrices.
- Single qubit and CNOT gates together can be used to implement an arbitrary two-level unitary operation on the state space of n qubits.
- Statement 3: Hadamard and the $\pi/8$ gate can be used to approximate any single-qubit unitary operation to arbitrary accuracy.

Think of a 2-level unitary as a 2x2 unitary matrix or a single-qubit operation.

Universal Decomposition

We want to express an algorithm U in terms of 2-level unitaries

$$U = \prod_i A_i, \text{ where } A_i \text{ are 2-level unitaries.}$$

Reduce one element at a time $UA_0 = U_1$:

$$\begin{bmatrix} \cdots & a & b & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} A_i = \begin{bmatrix} \cdots & c & 0 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \Rightarrow \begin{bmatrix} a & b \end{bmatrix} A_0' = \begin{bmatrix} c & 0 \end{bmatrix}$$

Reduce rows until left with 2x2:

$$\begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \to \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix} \to \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{bmatrix}$$

This procedure effectively breaks d-dimensional U into $\approx d^2 = (2^n)^2 = 4^n$ single-qubit operations A_i.

Solovay-Kitaev theorem

The Solovay-Kitaev theorem says that we can approximate an arbitrary quantum algorithm U using only gates from an instruction set G to within a desired error margin ϵ with a compiled sequence that has length poly-logarithmic to the error.

In short, we are guaranteed to find an efficient machine runnable sequence for any arbitrary quantum algorithm.

Theorem 1 (Solovay-Kitaev)

Let G be an instruction set for $SU(d)$, and $\epsilon > 0$ be a desired accuracy. There is a constant c such that for any $U \in SU(d)$, there exists a finite sequence S of gates from G of length $O(\log^c(1/\epsilon))$ and such that $d(U, S) < \epsilon$.

Solovay-Kitaev theorem

The initial **translation step** is to find a base approximation U_0 to U within error ϵ_0 using l_0 gates.

Through **recursive steps** of the **shrinking lemma**

$$l_1 \rightarrow 5l_0$$
$$\epsilon_1 \rightarrow \epsilon_0^{2/3}$$

we get better approx $U_2 \approx U$ with longer sequence obtained through the **group commutator** of U_1 and U_0.

Solovay-Kitaev theorem

Translation step

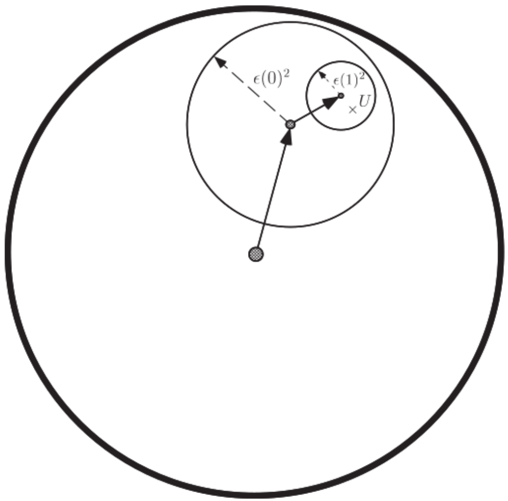

Figure. The translation step used in the proof of the Solovay-Kitaev theorem. To approximate single qubit gate we first approximate to within a distance $\epsilon(0)^2$ using l_0 gates from G. Then we improve the approximation by adding $5l_0$ more gates, for a total accuracy better than $\epsilon(1)^2$, and continue on this way, quickly converging to U.

Solovay-Kitaev theorem

Shrinking Lemma

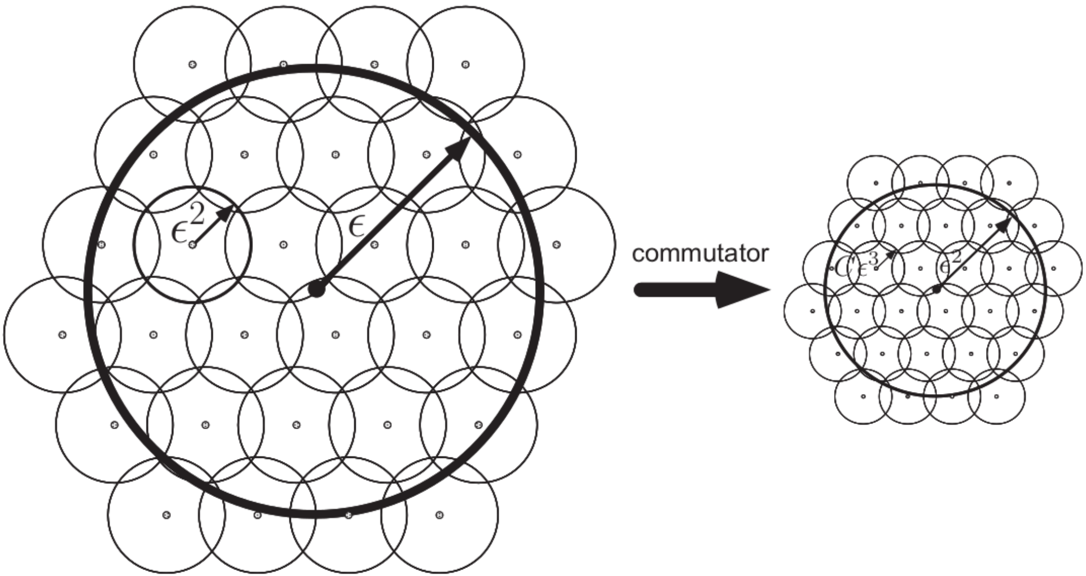

Figure. The main idea of the shrinking lemma. Taking group commutators of elements U_1 and U_2 dense in ϵ-net fills in ϵ^2-net much more densely function Solovay-Kitaev (Gate U , depth n)

if $(n == 0)$

  Return Basic Approximation to U

else

  Set $U_{n-1} =$ Solovay-Kitaev$(U, n-1)$

  $WVW^\dagger V^\dagger =$ **GCDecompose**$(UU_{n-1}^\dagger)$

  Set $V_{n-1} =$ Solovay-Kitaev$(V, n-1)$

  Set $W_{n-1} =$ Solovay-Kitaev$(W, n-1)$

Return $U_n = V_{n-1}W_{n-1}V_{n-1}^\dagger W_{n-1}^\dagger U_{n-1}$

At each recurrence: the error ϵ, the gate depth l increases, and the runtime t changes by:

$$\begin{aligned} \epsilon_d &= c_{\text{approx}}\, \epsilon_{d-1}^{3/2} \\ l_d &= 5 l_{d-1} \\ t_d &\leq 3 t_{d-1} + \text{ const.} \end{aligned}$$

By recursive relations, we can establish the complexity bounds for the error, gate depth, and runtime for the Solovay-Kitave algorithm by:

$$\begin{aligned} \epsilon_d &= \frac{1}{c_{\text{approx}}^2} \left(\epsilon_0 c_{\text{approx}}^2 \right)^{\left(\frac{3}{2}\right)^d} \\ l_d &= O\left(5^d\right) \\ t_d &= O\left(3^d\right) \end{aligned}$$

Implementation of Compiler

Universal Decomposition turns an algorithm U into a product of single-qubit gates.

The Solovay-Kitaev algorithm converts any single-qubit gates into a sequence of gates from hardware gate set G.

Combining Universal Decomposition and Solovay-Kitaev algorithm we have a compiler!

IMPLEMENTATION OF COMPILER

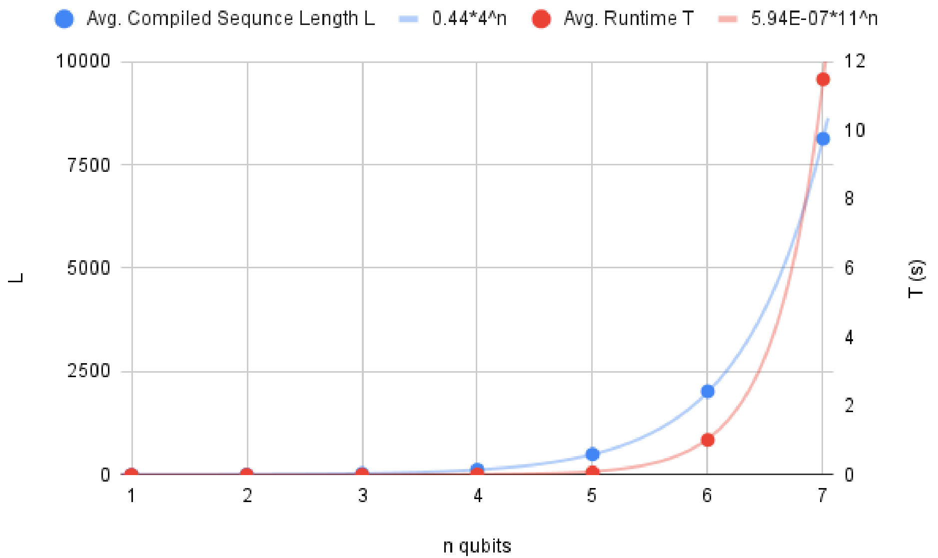

Figure. The Average Compiled Sequence Length and Average Runtime of Universal Decomposition versus the number of qubits n that defines the size of the unitary operator state space. Dots indicate data points and curves the best fit lines.

Implementation of Compiler

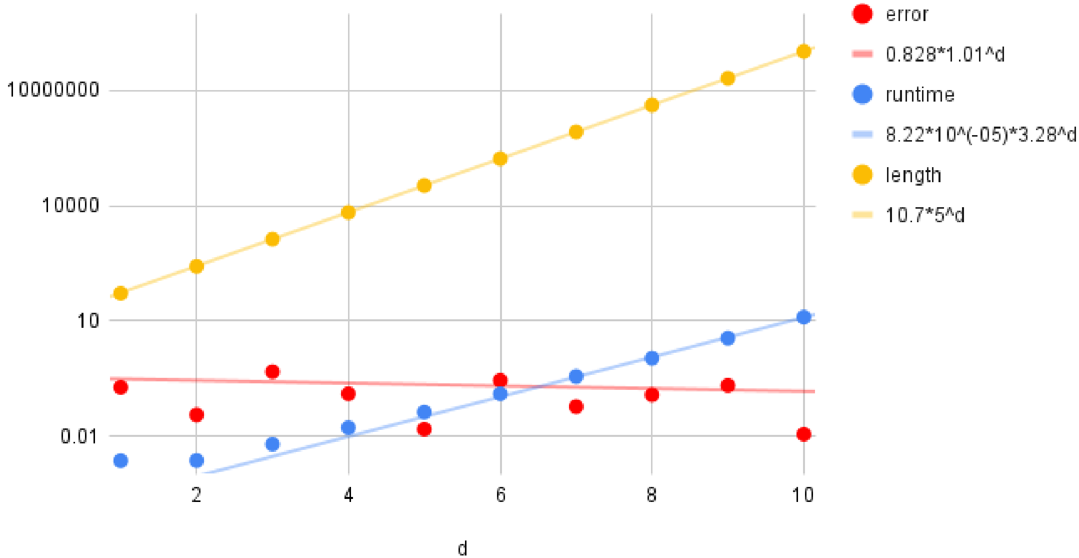

Figure. The log plot of average error ϵ_d, runtime t_d, and output compiled sequence length l_d over the recursive depth d of the Solovay-Kitaev algorithm.

IMPLEMENTATION OF COMPILER

Thank you for listening! Questions?