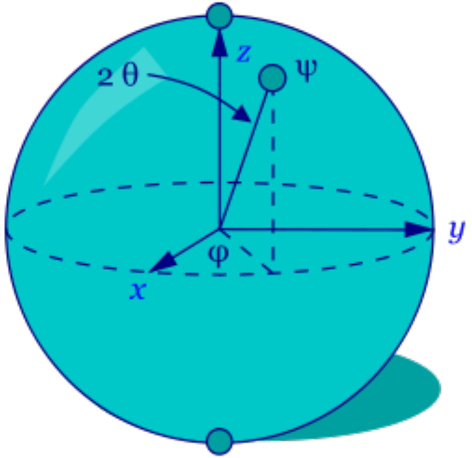


QuantumLib:

A Library for Quantum Computing in Coq

Collaborators

Jacob Zweifler
University of Chicago
jzweifler@uchicago.edu

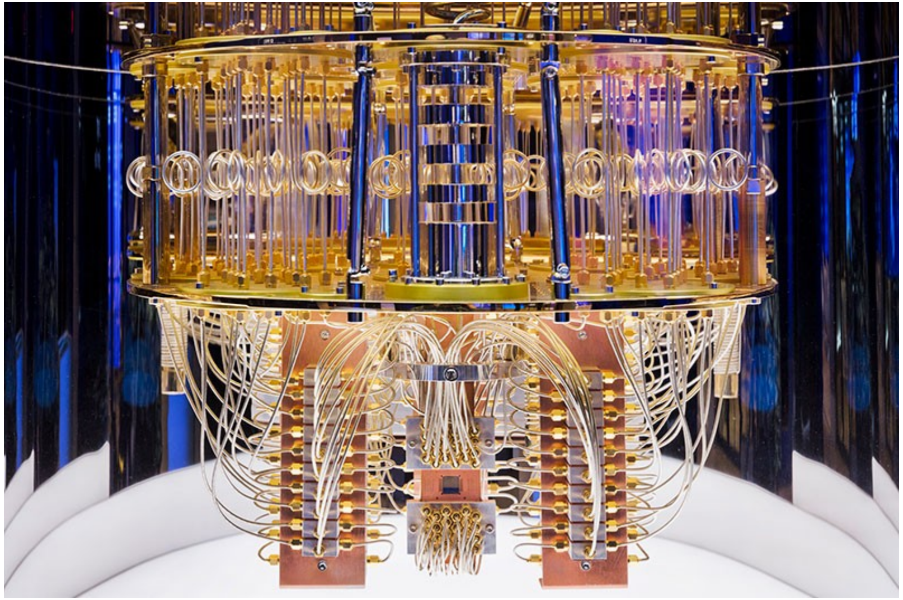
Kesha Hietala
Amazon
kesha@cs.umd.edu

Robert Rand
University of Chicago
rand@uchicago.edu

Additional list of contributors can be found at: https://github.com/inQWIRE/QuantumLib

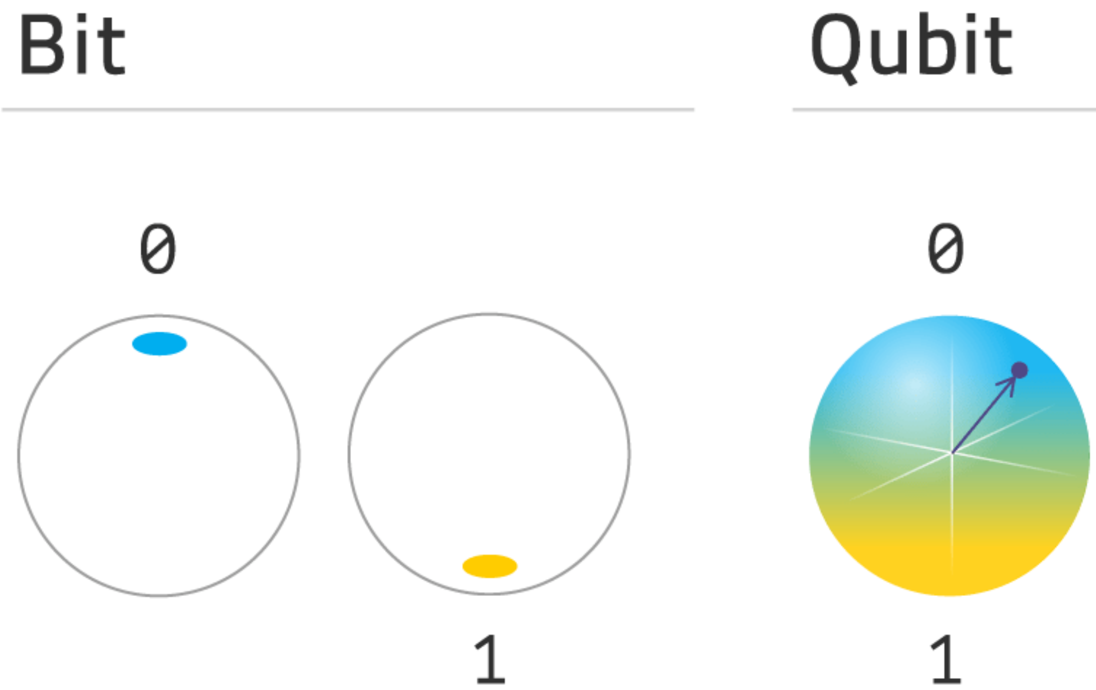
Why Verify Quantum Computing?

- Can be faster than classical systems
  - Quantum simulation
  - Shor's algorithm: encryption
  - Grover's search algorithm
- Quantum Advantage in practice
  - Google's random circuit sampling
  - Boson sampling (USTC, Xanadu)
- Quantum computing is hard!
  - Conceptually hard
  - Very error-prone



IBM's 127 qubit quantum computer

Quantum Bits: Qubits



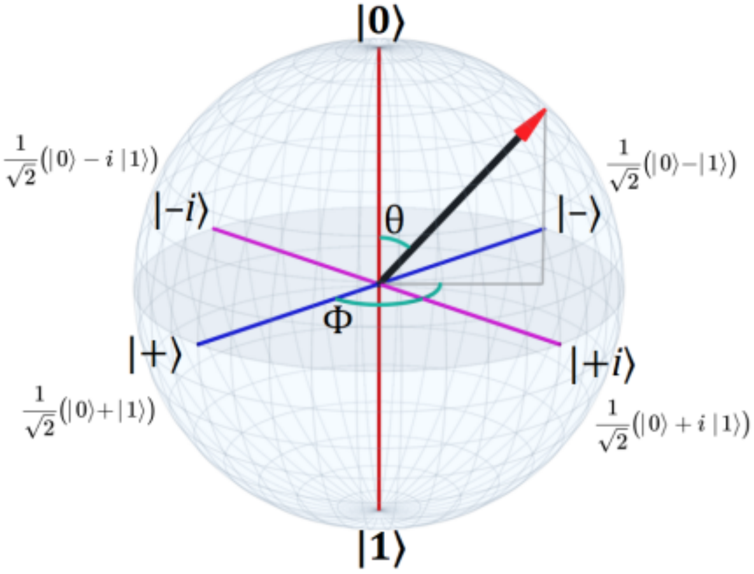
$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

Represented by $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$,

where $|\alpha|^2 + |\beta|^2 = 1$

Quantum Gates: Unitary Operators

- Gates act on qubits to change their state
  - Eg: X, Y, Z, H, S, T



Examples of gate application:

Corresponding matrix multiplication:

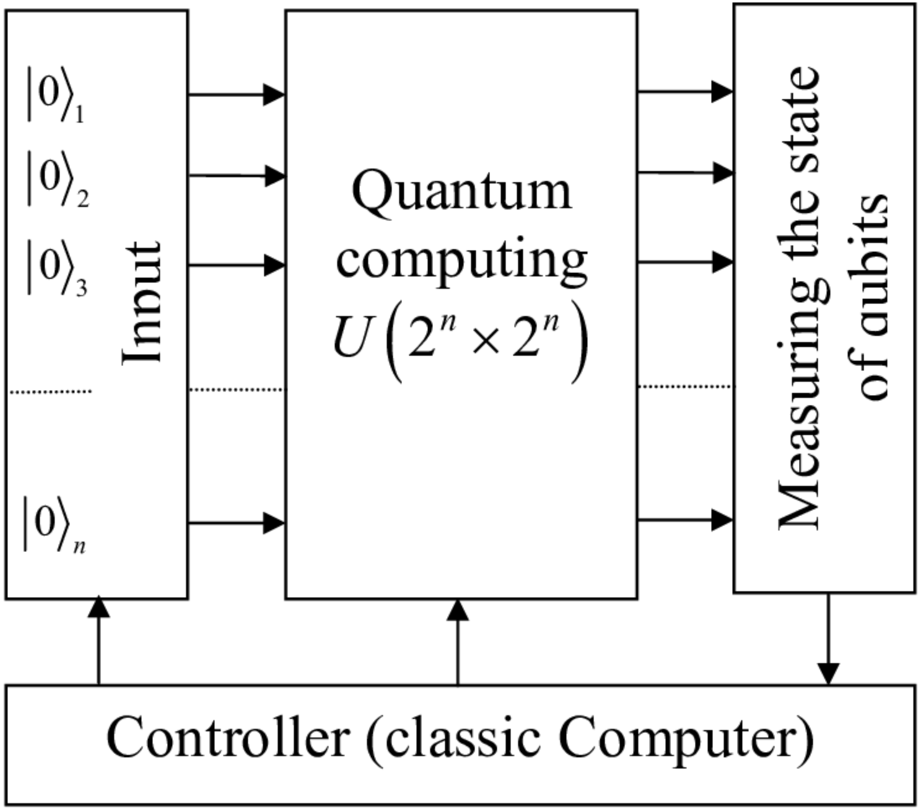
$$X\,|0\rangle = |1\rangle \longrightarrow \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$H\,|+\rangle = |0\rangle \longrightarrow \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Quantum Circuits

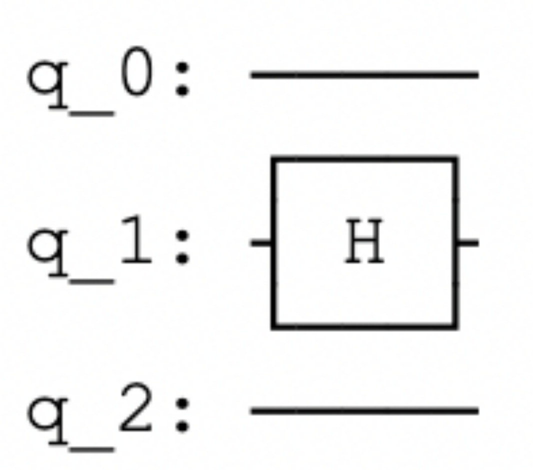
- Circuits represented by kronecker product:

$$\begin{pmatrix} \alpha_1 \\ \beta_1 \end{pmatrix} \otimes \ldots \otimes \begin{pmatrix} \alpha_n \\ \beta_n \end{pmatrix}$$



Quantum Programs

- Applying gates corresponds to multiplication by padded matrix:



$$\mathrm{I} \otimes \mathrm{H} \otimes \mathrm{I}$$

Applied to

$$\begin{bmatrix}\alpha_1 \\ \beta_1\end{bmatrix} \otimes \begin{bmatrix}\alpha_2 \\ \beta_2\end{bmatrix} \otimes \begin{bmatrix}\alpha_3 \\ \beta_3\end{bmatrix}$$

$$(\mathrm{I} \otimes \mathrm{H} \otimes \mathrm{I}) \times \left(\begin{bmatrix}\alpha_1 \\ \beta_1\end{bmatrix} \otimes \begin{bmatrix}\alpha_1 \\ \beta_1\end{bmatrix} \otimes \begin{bmatrix}\alpha_1 \\ \beta_1\end{bmatrix} \right) \longrightarrow \left(\mathrm{I} \times \begin{pmatrix}\alpha_1 \\ \beta_1\end{pmatrix} \right) \otimes \left(\mathrm{H} \times \begin{pmatrix}\alpha_2 \\ \beta_2\end{pmatrix} \right) \otimes \left(\mathrm{I} \times \begin{pmatrix}\alpha_3 \\ \beta_3\end{pmatrix} \right)$$

Why QuantumLib?

- Provide a backbone for quantum computing projects in Coq
- Tailored specifically towards quantum computing
  - QuantumLib is more efficient and comprehensive than other more general libraries
  - Can act as an extension of MathComp or Ccorn
- Consists of both low level and high level components

To do this, we rigorize the notions of Hilbert spaces in Coq: $\mathbb{C}^{2^n}$

Underlying Field Structure: Complex Numbers

- Coquelicot's complex numbers:
  - Added lemmas involving Euler's identity
- Polar coordinates
  - Eg: e^{ix}
- Summation notation
- Computable for our purposes
  - Sufficient rewrite lemmas
  - Proof that $\mathbb{C}$ is a field
- Polynomials over $\mathbb{C}$ and proof of completeness
  - Used for facts about determinants and eigenvectors

```
Definition C := (R * R)%type.
```

$$(AB)_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

Matrices Over ℂ

- Matrices defined as follows:

```
Definition Matrix (m n : nat) := nat -> nat -> C.
```

- Matrices must be well-formed:

```
Definition WF_Matrix {m n: nat} (A : Matrix m n) : Prop :=
  forall x y, x >= m \/ y >= n -> A x y = 0.
```

- Examples:

```
Definition σx : Matrix 2 2 :=
  fun x y => match x, y with
             | 0, 1 => 1
             | 1, 0 => 1
             | _, _ => 0
             end.
```

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

```
Definition I (n : nat) : Square n :=
  fun x y => if (x =? y) && (x <? n)
             then 1
             else 0.
```

Some Example Operations

```
Definition Mplus {m n : nat} (A B : Matrix m n) : Matrix m n :=
  fun x y => (A x y + B x y).
```

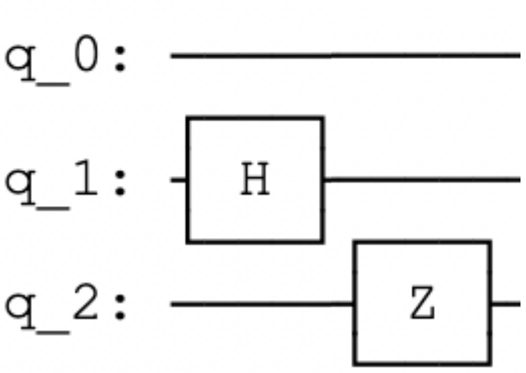
```
Definition Mmult {m n o : nat} (A : Matrix m n) (B : Matrix n o) : Matrix m o :=
  fun x z => big_sum (fun y => A x y * B y z) n.
```

Applying circuits in series

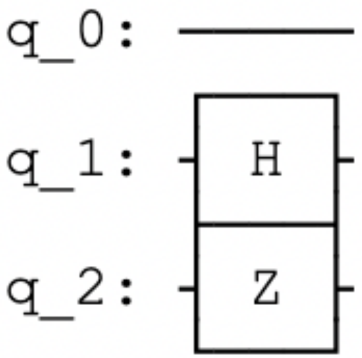
```
Definition kron {m n o p : nat} (A : Matrix m n) (B : Matrix o p) :
  Matrix (m*o) (n*p) :=
  fun x y => (A (x / o) (y / p)) * (B (x mod o) (y mod p)).
```

Applying circuits in parallel

Compatibility between Mmult and kron



=



Applying circuits in series

Applying circuits in parallel

```
Lemma program_equivalence : (I ⊗ H ⊗ I) × (I ⊗ I ⊗ Z) = (I ⊗ H ⊗ Z).
```

Why These Design Choices?

- Phantom types help with proofs
  - Useful since kron changes size of matrix

```
Lemma kron_assoc : forall {m n p q r s : nat}
  (A : Matrix m n) (B : Matrix p q) (C : Matrix r s),
  (A ⊗ B) ⊗ C = A ⊗ (B ⊗ C).
```

```
Matrix ((m*p)*r) ((n*q)*s)        Matrix (m*(p*r)) (n*(q*s))
```

```
Definition pad {n} (start dim : nat) (A : Square (2^n)) : Square (2^dim) :=
  if start + n <=? dim then I (2^start) ⊗ A ⊗ I (2^(dim - (start + n))) else Zero.
```

- Not clear that $(2^{start} * 2^{n} * 2^{dim - (start + n)}) = 2^{dim}$
  - Relies on guard

Other components

- Comprehensive linear algebra
  - Linear independence, diagonalizability, determinant
  - Eigenvectors
- Quantum components
  - Low level: basic states and gates
  - High level: padded gates, pure/mixed states
- Measurement and probability theory
- Translation to/from different representations of quantum objects
  - Vector states
  - Permutations

```
Lemma H0_spec : hadamard × |0⟩ = |+⟩.
Proof. lma'. Qed.

Lemma H1_spec : hadamard × |1⟩ = |−⟩.
Proof. lma'. Qed.
```

```
Definition pad_u (dim n : nat) (u : Square 2) :
  Square (2^dim) := @pad 1 n dim u.
```

```
Fixpoint f_to_vec (n : nat) (f : nat -> bool) : Vector (2^n) :=
  match n with
  | 0 => I 1
  | S n' =>  (f_to_vec n' f) ⊗ ∣ f n' ⟩
  end.
```

Matrix Tactics

- Well-formedness tactics
  - show_wf and wf_db
- lma
- gridify
  - Good for symbolic proofs
- solve_matrix
  - Good for numerical computations
- restore_dims

lma:

- Breaks down equality statement into cells
- Eg:

```
Lemma MmultYY : σy × σy = I 2. Proof. lma. Qed.
```

$$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \longrightarrow$$

0*0+(-i)*i	0*(-i)+(-i)*0
i*0+0*i	i*(-i)+0*0

...

gridify:

```
Lemma pad_mult : forall n dim start (A B : Square (2^n)),
  pad start dim A × pad start dim B = pad start dim (A × B).
Proof.
  intros.
  unfold pad.
  gridify.
  reflexivity.
Qed.
```

```
(I (2 ^ start) ⊗ A ⊗ I (2 ^ d)) × (I (2 ^ start) ⊗ B ⊗ I (2 ^ d))
=
I (2 ^ start) ⊗ (A × B) ⊗ I (2 ^ d)
```

```
Lemma pad_A_B_commutes : forall dim m n A B,
  m <> n ->
  WF_Matrix A ->
  WF_Matrix B ->
  pad_u dim m A × pad_u dim n B = pad_u dim n B × pad_u dim m A.
Proof.
  intros.
  unfold pad_u, pad.
  gridify; trivial.
Qed.
```

QuantumLib in Practice

- QWIRE
  - Quantum language built in QWIRE
    - Dependent types
- SQIR/VOQC
  - Uses translations between matrices
  - Proof of Shor’s algorithm, Grover’s algorithm, teleportation
- VyZX
  - ZX calculus: graphical representation of quantum computing
- Verification of Gottesman logic

Why QuantumLib Instead of Other Libraries?

- Works well with Kronecker product
- Tailored specifically to quantum computing
    - Contains additional components that other libraries lack
- Large math foundation
- Many rewrite lemmas/tactics
- Very compatible with other libraries (coming soon)

Challenges

- Slow when matrices get large
  - 3 qubits much worse than 2
  - 4 qubits very slow
  - Multiplication: $O(n^3)$ when n is matrix size so $O(2^{3n})$ when n is number of qubits
- Showing dimensions align
  - kron mixed product rule can be hard to apply
- Not actually computable
  - Leads to more slowness

Things for the Future

- Make tactics even better
  - Improve gridify by having different versions
- Add more linear algebra proofs
  - More matrix properties
  - Proof of Gottesman-Knill theorem
- Add more lemmas to Pad.v
  - More commutation lemmas
- Version with Ccorn for more computable matrices

More Things for the Future

- Group, ring, field typeclasses
  - field and ring exist in Coq, why not other structures?
    - Reification-style tactics
  - Generalize matrices to be over any field
  - Would make integration with Ccorn very easy
  - Finite fields

Summary

- Quantum computing occurs over Hilbert spaces
  - Strong interplay between matrix multiplication and kronecker product
- Matrix definition: `Definition Matrix (m n : nat) := nat -> nat -> C.`
  - Matrices must also be well formed
- Defined many quantum computing notions
- Matrix tactics
- Project found at https://github.com/inQWIRE/QuantumLib
- To install:
  - `opam install coq-quantumlib`