

Introducing Qibo

Towards an agnostic toolbox for quantum simulation and hardware control

Stefano Carrazza

30th November 2021

INFN



Introduction

Introduction

From a practical point of view, we are moving towards new technologies, in particular **hardware accelerators**:

CPU



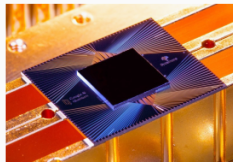
GPU



FPGA/ASIC



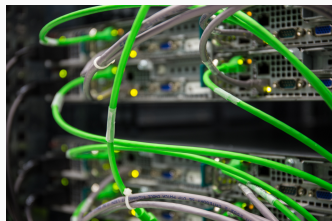
Quantum chip



Moving from **general purpose devices** \Rightarrow **application specific**

However, there are several challenges:

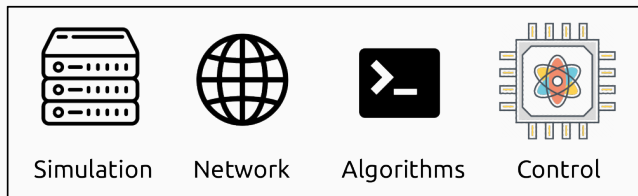
- simulate efficiently algorithms on classical hardware for QPU?
- control, send and retrieve results from the QPU?
- error mitigation, keep noise and decoherence under control?



How can we interact with QPU?

Solution:

Construct a Quantum Middleware:

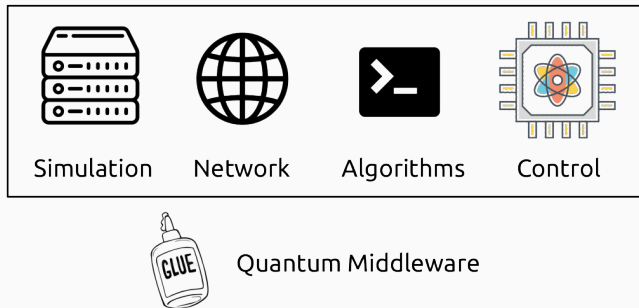


Quantum Middleware

How can we interact with QPU?

Solution:

Construct a Quantum Middleware:

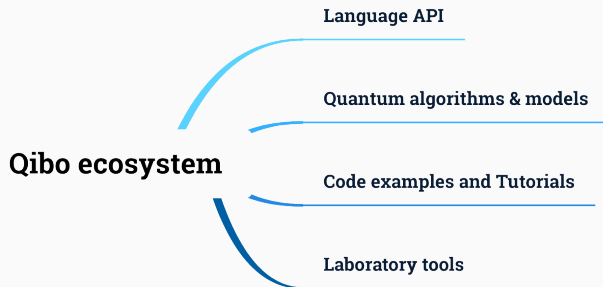


⇒ **Qibo: an open-source full-stack middleware.**

Introducing Qibo

Introducing Qibo

Qibo is an open-source full stack **API** for quantum simulation and hardware control. It is platform **agnostic** and supports **multiple backends**.

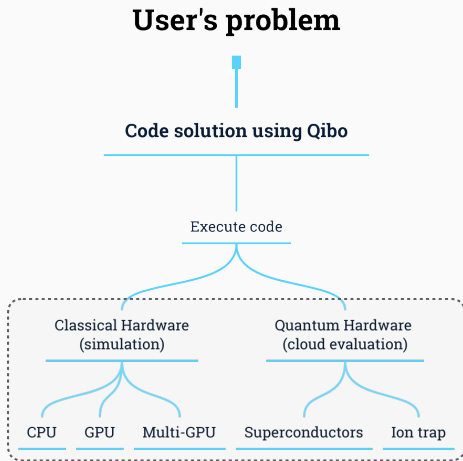


<https://github.com/qiboteam/qibo>

<https://arxiv.org/abs/2009.01845>

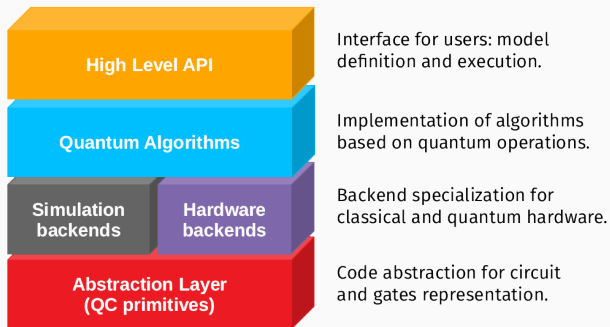


Abstractions in Qibo

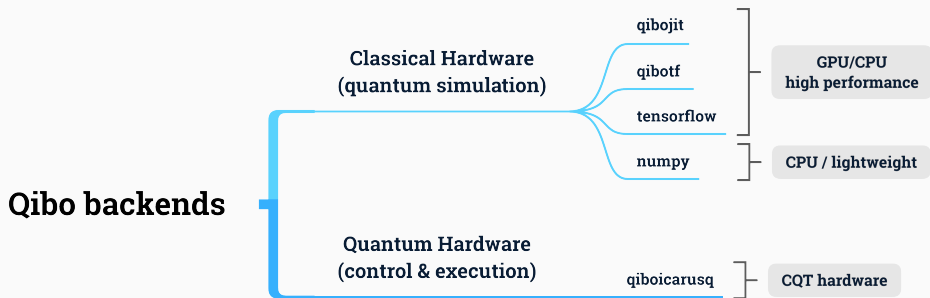


- Single piece of code
- Automatic deployment on simulators and quantum devices
- Plugin backends mechanism

Qibo Stack



Backends in Qibo



This layout opens the possibility to support:

- multiple classical and quantum hardware specifications
- hardware accelerators for simulation (single-GPU and multi-GPU)

numpy



`pip install qibo`

Simulator based on tensordot and linear algebra operations.

Features:

- Cross-architecture (x86, arm64, etc).
- Cross-platform.
- Fast for single-threaded operations.

tensorflow



`pip install tensorflow`

Simulator based on tensorflow primitives (einsum, matmul).

Features:

- Multithreading CPU.
- Single GPU.
- Gradient descent on quantum circuits.

qibotf



`pip install qibotf`

Simulator based on tensorflow custom operators in C++ and CUDA.

Features:

- Excellent single node performance.
- Multithreading CPU, single GPU and multi-GPU.
- Low memory footprint.

qibojit^{NEW}



`pip install qibojit`

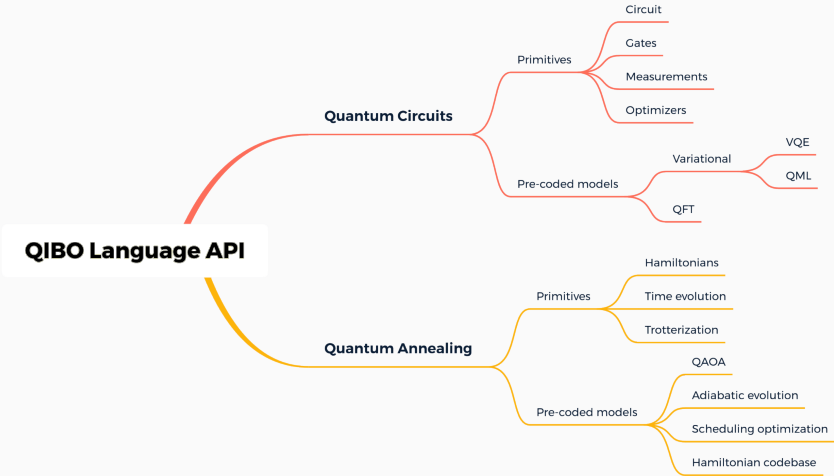
Simulator based on numba and cupy operations.

Features:

- Excellent single node performance.
- Multithreading CPU, single GPU and multi-GPU
- Cross-platform (just-in-time compilation)
- Works on NVIDIA and AMD GPUs.

Computational models in Qibo

Computational models in Qibo



Quantum Circuits

Quantum circuits

The **quantum circuit** model considers a sequence of unitary quantum gates:

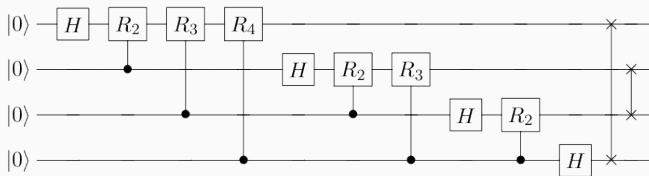
$$|\psi'\rangle = U_2 U_1 |\psi\rangle \quad \rightarrow \quad |\psi\rangle \text{ --- } \boxed{U_1} \text{ --- } \boxed{U_2} \text{ --- } |\psi'\rangle$$

Quantum circuits

The **quantum circuit** model considers a sequence of unitary quantum gates:

$$|\psi'\rangle = U_2 U_1 |\psi\rangle \quad \rightarrow \quad |\psi\rangle \text{ --- } \boxed{U_1} \text{ --- } \boxed{U_2} \text{ --- } |\psi'\rangle$$

For example a Quantum Fourier Transform with 4 qubits is represented by



Models based on Grover's algorithms and Shor's factorization algorithms.

Quantum gates












- **Single-qubit gates**

- Pauli gates
- Hadamard gate
- Phase shift gate
- Rotation gates

- **Two-qubit gates**

- Conditional gates
- Swap gate
- fSim gate

- Special gates: Toffoli

Operator	Gate(s)	Matrix
Pauli-X (X)	 	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)		$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S, P)		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$ (T)		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$
Controlled Not (CNOT, CX)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
Controlled Z (CZ)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
SWAP		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Toffoli (CCNOT, CCX, TOFF)		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

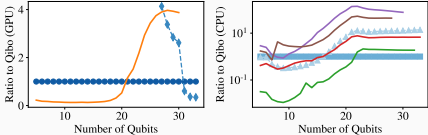
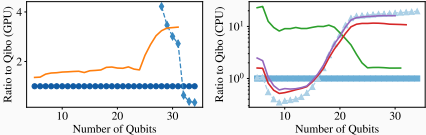
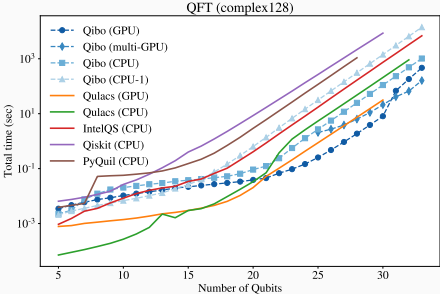
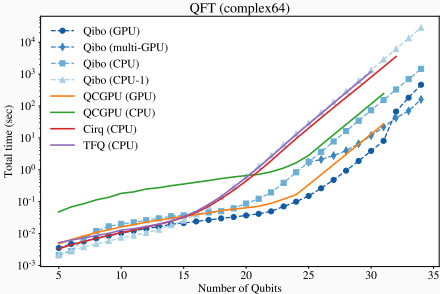
The final state of circuit evaluation is given by:

$$\psi'(\sigma_1, \dots, \sigma_N) = \sum_{\tau'} G(\tau, \tau') \psi(\sigma_1, \dots, \tau', \dots, \sigma_N),$$

where the sum runs over qubits targeted by the gate.

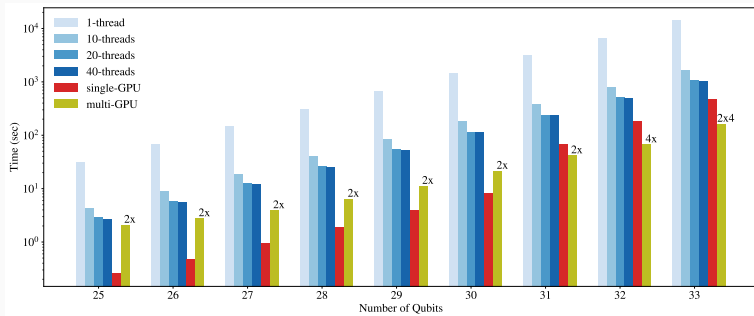
- Linear algebra approach.
- Possibility to parallelize and optimize operations.

Quantum circuit performance results



Quantum Fourier Transform performance.

Multi-GPU trade-off

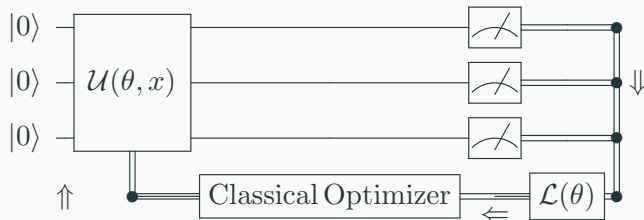


Quantum Fourier Transform performance.

Variational Quantum Circuits

Variational Quantum Circuits

Typical variational quantum circuits and data re-uploading algorithms:

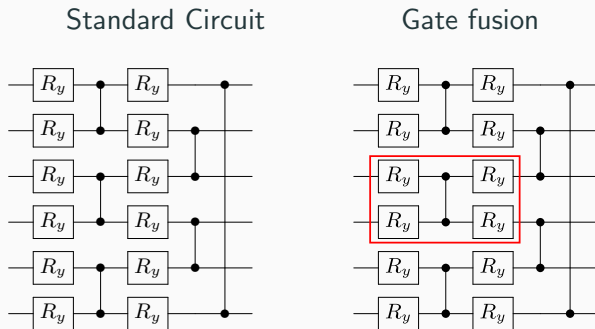


Define new parametric model architectures for quantum hardware:

\Rightarrow **Variational Quantum Circuits & Quantum Machine Learning**

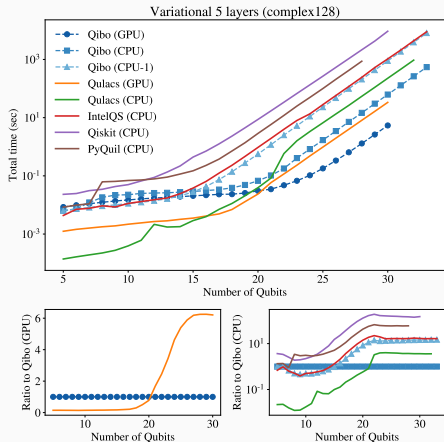
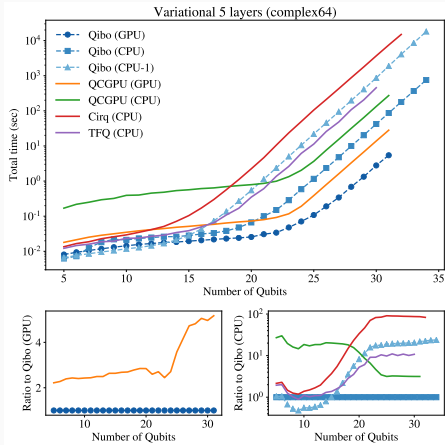
Variational circuit

Variational circuits are inspired by the structure of variational circuits used in **quantum machine learning**.



Qibo implements the **gate fusion** of four R_y and the controlled-phased gate, C_z
 \Rightarrow Qibo provides multi-qubit gate operators for CPU and GPU

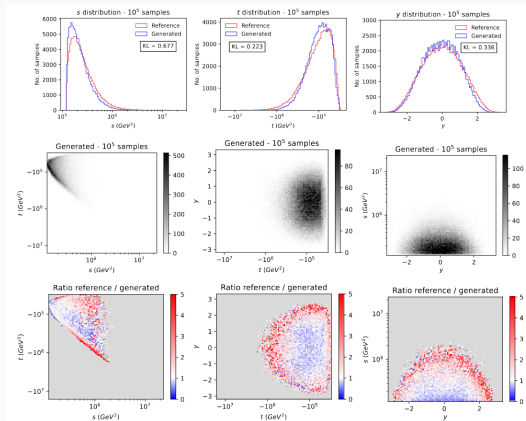
Benchmarks



Variational circuit simulation performance comparison in single and double precision.

Summary of circuit-based built-in models in Qibo

- Variational quantum eigensolver
- Quantum approximate optimization algorithm (QAOA)
- Feedback-based algorithm for quantum optimization (FALQON)
- Quantum Neural Networks
 - Variational quantum classifier
 - Variational quantum regressor
 - Style-based quantum GAN



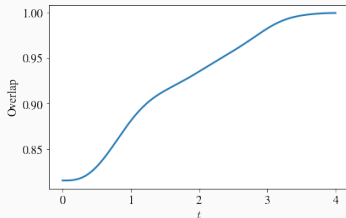
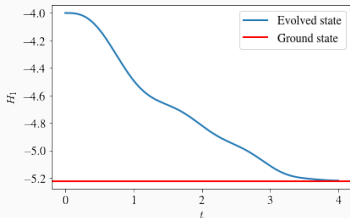
Quantum Annealing

- **Annealing quantum processors**

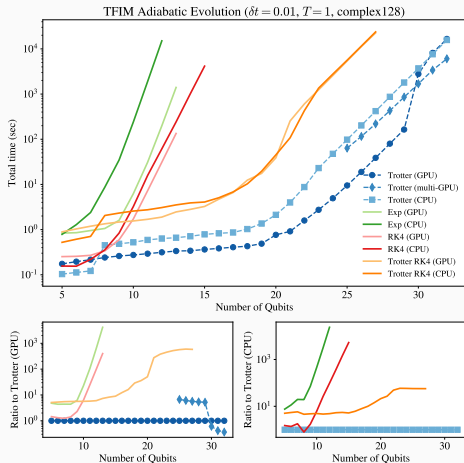
- Hamiltonian database
- Time evolution of quantum states
- Adiabatic Evolution simulation
- Scheduling determination
- Trotter decomposition

$$i\hbar \frac{\partial}{\partial t} |\psi(t)\rangle = H(s) |\psi(t)\rangle$$

$$H(t) = (1 - s(t))H_0 + s(t)H_1,$$



Adiabatic evolution



Adiabatic evolution performance using Qibo and TFIM for exact and Trotter solution.

Quantum hardware control

Software control for quantum hardware

Ideally, we would like to:

- 1 Define a circuit and/or algorithm.
- 2 Send and retrieve results from QPU:



Users

```
import numpy as np
from qibo import models, gates

# create a circuit for N=3 qubits
circuit = models.Circuit(3)

# add some gates in the circuit
circuit.add([gates.H(0), gates.X(1)])
circuit.add(gates.RX(0, theta=np.pi/6))

# execute the circuit and obtain the
# final state
final_state = circuit()
```

Circuit using Qibo

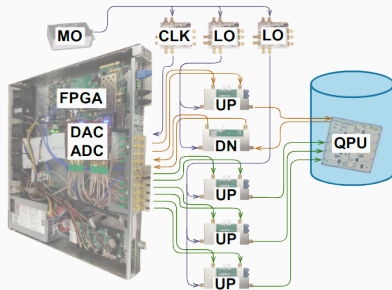


QPU

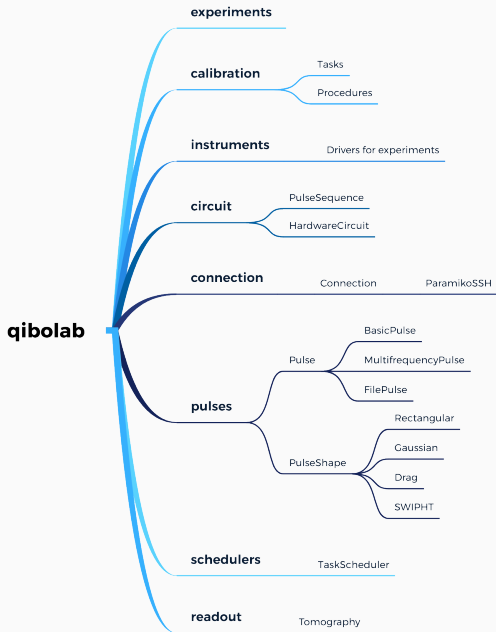
Software control for quantum hardware

From a hardware perspective this requires:

- Convert circuit into **microwave pulse sequences**.
- Operate multiple **instruments and FPGAs**.
- Perform system **calibration** periodically.
- Schedule and execute operations.
- Reconstruct measurements.



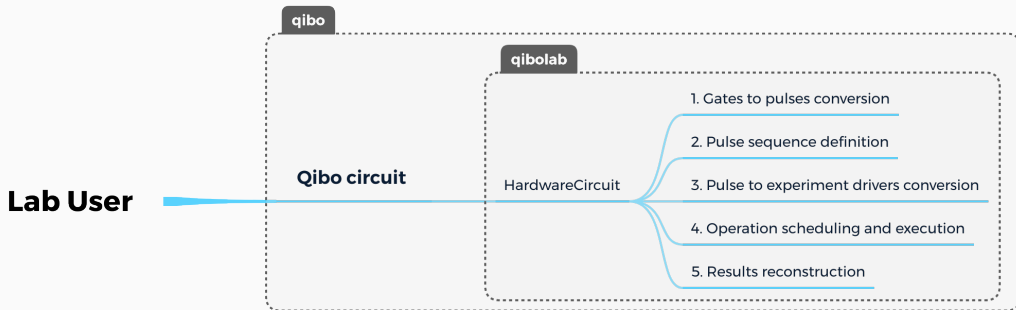
System layout from arXiv:2101.00071



QPU support using qibolab:

- Agnostic layout.
- Multiple experiments support.
- Plug & play for instruments.
- Tools for hardware control.

Qibo lab supported instruments



Qibo lab deployment



```
import numpy as np
from qibo.models import Circuit
from qibo import gates

c = Circuit(2)
c.add(gates.X(0))

# Add a measurement register on both qubits
c.add(gates.M(0, 1))

# Execute the circuit with the default initial state |00>.
result = c(nshots=100)
```



Lab Server

Qibo-Hardware backend
Circuit Compiler + pulse controller

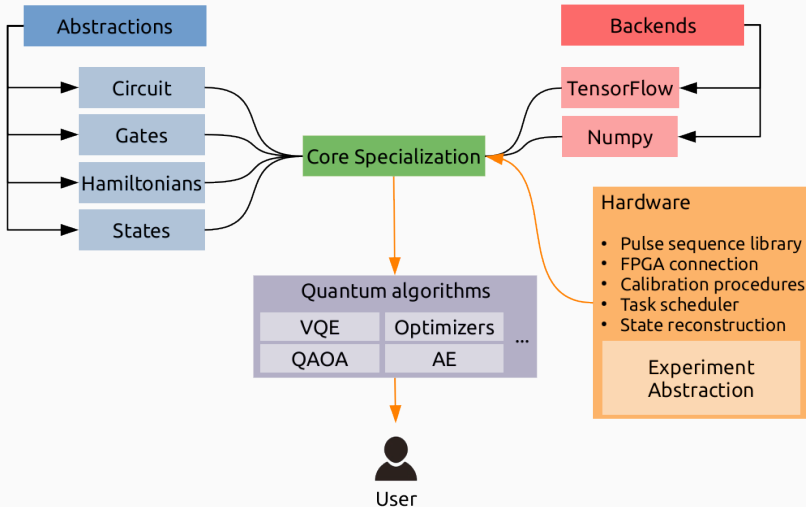
FPGA RFSoc



QPU



Qibo Layout with Hardware



Supported instruments

- AWG:
 - Tektronix (e.g. AWG5204, AWG70000A)
 - AlazarTech boards (e.g. ATS9371)
 - QuickSyn
 - Rigol (DC 5072)
 - ...
- QBlox*
- FPGA boards:
 - Xilinx Zynq UltraScale+ RFSoc*
 - Intel Cyclone V*

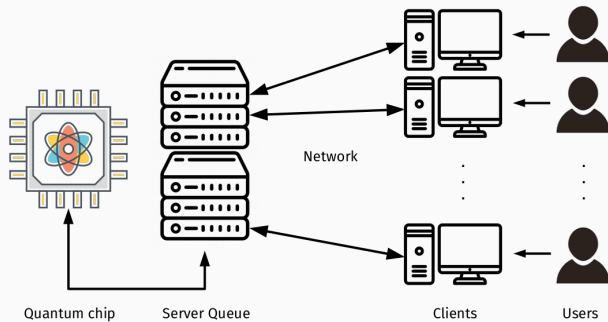
* supported system under development

Development roadmap:

- Qibo already provides a prototype approach based on AWG-like instruments.
- We are working on production control hardware based on FPGA boards.

Qibo - Server-client communication

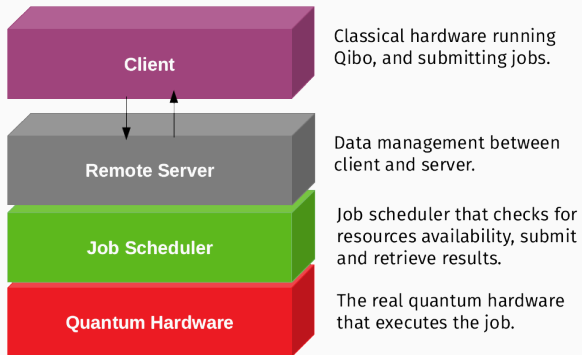
Remote access to QPU



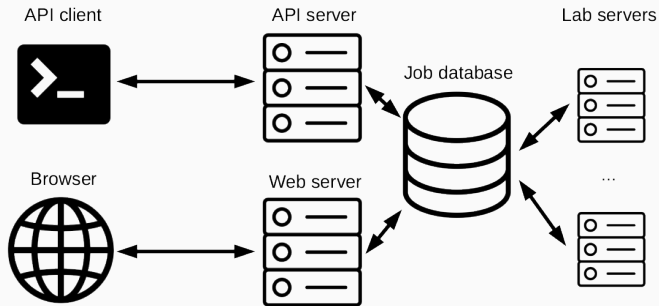
Goals:

- 1 Accept jobs from remote users (clients)
- 2 Schedule jobs in a server queue (server)
- 3 Run and retrieve results from QPU (server)

Qibo Middleware



Client-server infrastructure layout



Development of 2 modules:

- **Client:** translates circuits and algorithms into requests.
- **Server:** complete webserver and queue system for job submission.

Outlook

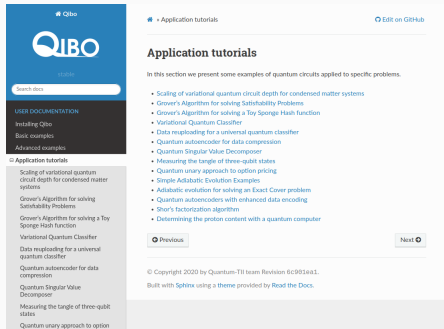
Qibo is currently a framework for research:

- 1 publicly available as an open-source code:
<https://github.com/qiboteam/qibo>
- 2 Designed with several **abstraction** layers.
- 3 For **fast prototyping** of quantum algorithms.

The screenshot shows the GitHub repository for Qibo. At the top, there's a navigation bar with links for 'Why QIBO?', 'Team', 'Enterprise', 'Custom', 'Marketplace', and 'Pricing'. Below that, the repository name 'qiboteam/qibo' is displayed. The main content area shows a list of recent pull requests, including 'Merge pull request #151 from qiboteam/feature/...', 'qibo', 'doc', 'examples', 'uvloop', 'qiboterm', 'qiboapi', 'addthedocs.yml', 'LICENSE', 'README.md', 'qiboctl', 'requirements.txt', and 'setup.py'. Below the pull requests, there's a section for the QIBO logo and a description: 'Qibo is an open source full stack API for quantum simulation and quantum hardware control. Some of the key features of Qibo are:'. A list of features follows, including 'Definition of a standard language for the construction and execution of quantum circuits with device agnostic approach to simulation and quantum hardware control based on plug and play backend drivers', 'A continuously growing code-base of quantum algorithms applications presented with easegates and tutorials', 'Efficient simulation backends with GPU, multi-GPU and CPU with multi-threading support', and 'Simple mechanism for the implementation of new simulation and hardware backend drivers'. On the right side, there's an 'About' section, 'Releases' (showing Qibo 1.1.0), 'Packages', 'Contributors', and 'Languages'.

We provide several tutorials for:

- Variational circuits
- Grover's algorithm
- Adiabatic evolution
- Quantum Singular Value Decomposer
- ...



The screenshot shows the Qibo website interface. At the top, there is a blue header with the QIBO logo and a search bar. Below the header, there is a dark navigation menu with options like 'Installing Qibo', 'Basic examples', and 'Advanced examples'. The main content area is titled 'Application tutorials' and lists various quantum algorithms and applications. A 'Previous' button is visible at the bottom of the page.

Qibo

SEARCH

Search docs

USER DOCUMENTATION

Installing Qibo

Basic examples

Advanced examples

Application tutorials

- Scaling of variational quantum circuit depth for condensed matter systems
- Grover's Algorithm for solving Satisfiability Problems
- Grover's Algorithm for solving a Toy Sponge Hash function
- Variational Quantum Classifier
- Data reencoding for a universal quantum classifier
- Quantum autoencoder for data compression
- Quantum autoencoder for data compression
- Quantum Singular Value Decomposer
- Measuring the tangle of three-qubit states
- Quantum unary approach to option pricing

Application tutorials Edit on GitHub

In this section we present some examples of quantum circuits applied to specific problems.

- Scaling of variational quantum circuit depth for condensed matter systems
- Grover's Algorithm for solving Satisfiability Problems
- Grover's Algorithm for solving a Toy Sponge Hash function
- Variational Quantum Classifier
- Data reencoding for a universal quantum classifier
- Quantum autoencoder for data compression
- Quantum Singular Value Decomposer
- Measuring the tangle of three-qubit states
- Quantum unary approach to option pricing
- Simple Adiabatic Evolution Examples
- Adiabatic evolution for solving an Exact Cover problem
- Quantum autoencoders with enhanced data encoding
- Shor's factorization algorithm
- Determining the proton content with a quantum computer

Previous Next

© Copyright 2020 by Quantum-TIL team. Revision 6c9034a1.
Built with Sphinx using a theme provided by Read the Docs.

Visit:

<https://qibo.readthedocs.io/en/stable/code-examples/applications.html>

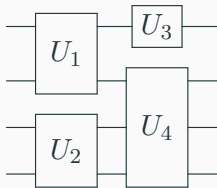
Thank you for your attention.

Rational for Variational Quantum Circuits

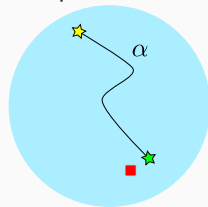
Rational:

Deliver variational quantum states \rightarrow explore a large Hilbert space.

$$U(\vec{\alpha}) = U_n \dots U_2 U_1$$



Near optimal solution

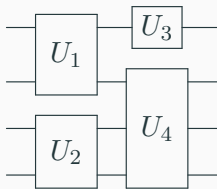


Rational for Variational Quantum Circuits

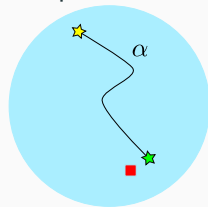
Rational:

Deliver variational quantum states \rightarrow explore a large Hilbert space.

$$U(\vec{\alpha}) = U_n \dots U_2 U_1$$



Near optimal solution



Idea:

Quantum Computer is a machine that generates variational states.

\Rightarrow **Variational Quantum Computer!**

Solovay-Kitaev Theorem

Let $\{U_i\}$ be a dense set of unitaries.

Define a circuit approximation to V :

$$|U_k \dots U_2 U_1 - V| < \delta$$

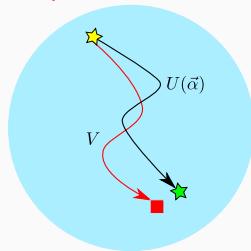
Scaling to best approximation

$$k \sim \mathcal{O}\left(\log^c \frac{1}{\delta}\right)$$

where $c < 4$.

⇒ The approximation is **efficient** and requires a **finite number of gates**.

Optimal solution



Adiabatic evolution

Example for adiabatic quantum computation:

Lets consider the evolution Hamiltonian:

$$H(t) = (1 - s(t))H_0 + s(t)H_1,$$

where

- H_0 is a Hamiltonian whose ground state is easy to prepare and is used as the initial condition,
- H_1 is a Hamiltonian whose ground state is hard to prepare
- $s(t)$ is a scheduling function.

According to the adiabatic theorem, for proper choice of $s(t)$ and total evolution time T , the final state $|\psi(T)\rangle$ will approximate the ground state of the “hard” Hamiltonian H_1 .