Introducing Qibo

from quantum circuits to machine learning arXiv:2009.01845

Stefano Carrazza
29th November 2021

ACAT 2021
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:

- Simulation
- Network
- Algorithms
- Control

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.

User's problem

Code solution using Qibo

Execute code

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

Qibo Stack

- **High Level API**: Interface for users: model definition and execution.
- **Quantum Algorithms**: Implementation of algorithms based on quantum operations.
- **Simulation backends**
- **Hardware backends**: Backend specialization for classical and quantum hardware.
- **Abstraction Layer (QC primitives)**: Code abstraction for circuit and gates representation.
This layout opens the possibility to support:

- multiple classical and quantum hardware specifications
- hardware accelerators for simulation (single-GPU and multi-GPU)
<table>
<thead>
<tr>
<th><strong>numpy</strong></th>
<th><strong>tensorflow</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>pip install qibo</td>
<td>pip install tensorflow</td>
</tr>
<tr>
<td>Simulator based on tensordot and linear algebra operations.</td>
<td>Simulator based on tensorflow primitives (einsum, matmul).</td>
</tr>
<tr>
<td><strong>Features:</strong></td>
<td><strong>Features:</strong></td>
</tr>
<tr>
<td>• Cross-architecture (x86, arm64, etc).</td>
<td>• Multithreading CPU.</td>
</tr>
<tr>
<td>• Cross-platform.</td>
<td>• Single GPU.</td>
</tr>
<tr>
<td>• Fast for single-threaded operations.</td>
<td>• Gradient descent on quantum circuits.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>qibotf</strong></th>
<th><strong>qibojit</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>pip install qibotf</td>
<td>pip install qibojit</td>
</tr>
<tr>
<td>Simulator based on tensorflow custom operators in C++ and CUDA.</td>
<td>Simulator based on numba and cupy operations.</td>
</tr>
<tr>
<td><strong>Features:</strong></td>
<td><strong>Features:</strong></td>
</tr>
<tr>
<td>• Excellent single node performance.</td>
<td>• Excellent single node performance.</td>
</tr>
<tr>
<td>• Multithreading CPU.</td>
<td>• Multithreading CPU, single GPU and multi-GPU</td>
</tr>
<tr>
<td>• Multi-GPU.</td>
<td>• Cross-platform (just-in-time compilation)</td>
</tr>
<tr>
<td>• Low memory footprint.</td>
<td>• Works on NVIDIA and AMD GPUs.</td>
</tr>
</tbody>
</table>
Computational models in Qibo
Computational models in Qibo

QIBO Language API

Quantum Circuits
- Primitives
- Gates
- Measurements
- Optimizers
- Variational
- QFT
- Pre-coded models
- VQE
- QML

Quantum Annealing
- Hamiltonians
- Primitives
- Time evolution
- Trotterization
- QAOA
- Adiabatic evolution
- Scheduling optimization
- Hamiltonian codebase
- Pre-coded models
Quantum Circuits
The quantum circuit model considers a sequence of unitary quantum gates:

\[ |\psi'\rangle = U_2 U_1 |\psi\rangle \rightarrow |\psi\rangle \xrightarrow{U_1} U_2 |\psi'\rangle \]
Quantum circuits

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

$$|\psi'\rangle = U_2 U_1 |\psi\rangle \rightarrow |\psi\rangle \xrightarrow{U_1} U_2 |\psi'\rangle$$

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**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Gate(s)</th>
<th>Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pauli-X (X)</td>
<td><img src="image" alt="X gate" /></td>
<td>$\begin{bmatrix} 0 &amp; 1 \ 1 &amp; 0 \end{bmatrix}$</td>
</tr>
<tr>
<td>Pauli-Y (Y)</td>
<td><img src="image" alt="Y gate" /></td>
<td>$\begin{bmatrix} 0 &amp; -i \ i &amp; 0 \end{bmatrix}$</td>
</tr>
<tr>
<td>Pauli-Z (Z)</td>
<td><img src="image" alt="Z gate" /></td>
<td>$\begin{bmatrix} 1 &amp; 0 \ 0 &amp; -1 \end{bmatrix}$</td>
</tr>
<tr>
<td>Hadamard (H)</td>
<td><img src="image" alt="H gate" /></td>
<td>$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 &amp; 1 \ 1 &amp; -1 \end{bmatrix}$</td>
</tr>
<tr>
<td>Phase (S, P)</td>
<td><img src="image" alt="S gate" /></td>
<td>$\begin{bmatrix} 1 &amp; 0 \ 0 &amp; e^{i\pi/4} \end{bmatrix}$</td>
</tr>
<tr>
<td>$\pi/8$ (T)</td>
<td><img src="image" alt="T gate" /></td>
<td>$\begin{bmatrix} 1 &amp; e^{i\pi/4} \ 0 &amp; 0 \end{bmatrix}$</td>
</tr>
<tr>
<td>Controlled Not (CNOT, CX)</td>
<td><img src="image" alt="CNOT gate" /></td>
<td>$\begin{bmatrix} 1 &amp; 0 &amp; 0 &amp; 0 \ 0 &amp; 1 &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; 1 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 1 \end{bmatrix}$</td>
</tr>
<tr>
<td>Controlled Z (CZ)</td>
<td><img src="image" alt="CZ gate" /></td>
<td>$\begin{bmatrix} 1 &amp; 0 &amp; 0 &amp; 0 \ 0 &amp; 1 &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; 1 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; -1 \end{bmatrix}$</td>
</tr>
<tr>
<td>SWAP</td>
<td><img src="image" alt="SWAP gate" /></td>
<td>$\begin{bmatrix} 1 &amp; 0 &amp; 0 &amp; 0 \ 0 &amp; 1 &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; 1 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 1 \end{bmatrix}$</td>
</tr>
<tr>
<td>Toffoli (CCNOT, CCX, TOFF)</td>
<td><img src="image" alt="Toffoli gate" /></td>
<td>$\begin{bmatrix} 1 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 \ 0 &amp; 1 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; 1 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 1 &amp; 0 &amp; 0 &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 0 &amp; 1 &amp; 0 &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 1 &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 1 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 1 \end{bmatrix}$</td>
</tr>
</tbody>
</table>
The final state of circuit evaluation is given by:

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

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.

<table>
<thead>
<tr>
<th>Number of Qubits</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>2x</td>
</tr>
<tr>
<td>26</td>
<td>2x</td>
</tr>
<tr>
<td>27</td>
<td>2x</td>
</tr>
<tr>
<td>28</td>
<td>2x</td>
</tr>
<tr>
<td>29</td>
<td>2x</td>
</tr>
<tr>
<td>30</td>
<td>2x</td>
</tr>
<tr>
<td>31</td>
<td>2x</td>
</tr>
<tr>
<td>32</td>
<td>4x</td>
</tr>
<tr>
<td>33</td>
<td>2x4</td>
</tr>
</tbody>
</table>

1-thread, 10-threads, 20-threads, 40-threads, single-GPU, multi-GPU.
Variational Quantum Circuits

Typical variational quantum circuits and data re-uploading algorithms:

\[ |0\rangle \xrightarrow{U(\theta,x)} |0\rangle \]

Define new parametric model architectures for quantum hardware:

⇒ Variational Quantum Circuits & Quantum Machine Learning
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, $C_z$.

⇒ Qibo provides multi-qubit gate operators for CPU and GPU.
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
Qibo features

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

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

\[
H(t) = (1 - s(t))H_0 + s(t)H_1,
\]
Adiabatic evolution performance using Qibo and TFIM for exact and Trotter solution.
Quantum hardware control
Ideally, we would like to:

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

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

circuit = models.Circuit(3)

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

final_state = circuit()
```

Users  Circuit using Qibo  QPU
However, from a hardware perspective this requires:

- Convert circuit into microwave pulse sequences.
- Operate multiple remote FPGA boards.
- Perform system calibration periodically.
- Schedule all operations.
- Reconstruct measurements (e.g. tomography).
- Store results / perform hybrid calculations.
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.
We provide several tutorials for:

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

Thank you for your attention.
Rational for Variational Quantum Circuits

Rational:

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

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

Near optimal solution
Rational for Variational Quantum Circuits

Rational:

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

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

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 \( V \):

\[
|U_k \ldots U_2 U_1 - V| < \delta
\]

Scaling to best approximation

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

where \( c < 4 \).

⇒ The approximation is efficient and requires a finite number of gates.
Example for adiabatic quantum computation:

Let's 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 $H_1$. 