CIRQ and PennyLane

Gabriella Bettonte and Sara Marzella

CINECA

QUANTUM COMPUTING LAB

QC Emulators

Most quantum hardware platforms that are currently being developed support a circuit model of quantum computation. It is thus of interest to the community of quantum algorithm researchers to have access to advanced computational tools that can help them in their research while quantum hardware is being developed. In addition to access to QPUs, quantum circuit simulators are of interest for a variety of reasons. First, it allows the researcher to prototype and investigate novel algorithms without using up valuable quantum resources for debugging and testing their ideas. This can instead all be done using simulations on classical hardware. Second, as the currently available quantum hardware is still suffering from noise, it is not possible to study algorithms that require a circuit depth that lies beyond the coherence limits of the device. Many algorithms of interest, for example in molecular sciences [10], require deep circuits.

Ref: https://arxiv.org/pdf/2303.00123.pdf

- **Qiskit** with the Aer simulator backend, developed by **IBM**
- **PennyLane,** developed by **Xanadu**
- **Cirq** with the qsim simulator, developed by **Google Quantum AI**

CIRQ

Cirq

Cirq is a Python library for writing, manipulating, and optimizing quantum circuits and running them against quantum computers and simulators.

Try gpu_test.py and get_started.py!

CIRQ: GHZ state

In this example, we run a circuit that creates a Greenberger-Horne-Zeilinger (GHZ) state and samples experimental outcomes. The following Python script gets the amplitudes in `|0…00>` and `|1…11>` by calling three different simulators:

- Cirq built-in simulator
- qsim CPU-based simulator
- qsim accelerated with cuStateVec

```
(cirq) [gbettoni@login01 cirq]$ more output_1873672.out
cirq.sim   : [(0.7071067690849304+0j), (0.7071067690849304+0j)]
Wall Time: 145.4514925898984    -- Process Time: 144.96871383599998
qsim(CPU) : [(0.7071067690849304+0j), (0.7071067690849304+0j)]
Wall Time: 49.15862099383958    -- Process Time: 191.27186330600003
cuStateVec: [(0.7071067690849304+0j), (0.7071067690849304+0j)]
Wall Time: 2.9102182360365987    -- Process Time: 2.872959089999995
```

CIRQ: simulate a large circuit

Caution: Because of its size, this large circuit will not run on a default Google Colab kernel and is unlikely to run on your personal computer. Follow the Quantum simulation on GCP tutorial to run on a larger Google Cloud VM.

https://quantumai.google/qsim/tutorials/q32d14

This tutorial builds a large circuit—with 32 qubits and a gate depth of 14—in the following steps:

1. Build: The circuit build has two steps:
    - Layout the grid of qubits
    - Define the gate operations on the grid of qubits
2. Run options: In the final cell, the *qsim* simulator is run.
    - Set `options = {'t' = 16, 'v' = 3}` for 16 Cores,
    - and `Verbosity = 3`.
3. Run Calculate Amplitudes: The final step is running the circuit and calculating the amplitudes for two input bitstrings, as described by qsim_amplitudes usage.

Leonardo: ~3.5min, 4 GPUs

```
Result: [(-3.8697517084074 21e-06-5.218992555455 77e-06j), (7.610340162500506e-06+9.7049
Wall Time: 216.3622438579332    -- Process Time: 851.0485140740001
init time is 3.11507 seconds.
```

CINECA QUANTUM COMPUTING LAB

Pennylane

PennyLane is a **cross-platform Python library** for differentiable programming of quantum computers.

Key Features

- *Machine learning on quantum hardware.*
  Connect to quantum hardware using PyTorch, TensorFlow, JAX, Keras, or NumPy. Build rich and flexible hybrid quantum-classical models.
- *Device-independent.*
  Run the same quantum circuit on different quantum backends. Install plugins to access even more devices, including Strawberry Fields, Amazon Braket, IBM Q, Google Cirq, Rigetti Forest, Qulacs, Pasqal, Honeywell, and more.
- *Batteries included.*
  Built-in tools for quantum machine learning, optimization, and quantum chemistry. Rapidly prototype using built-in quantum simulators with backpropagation support.

Pennylane built in devices

default.qubit

A simple state-vector qubit simulator written in Python, with Autograd, JAX, TensorFlow, and Torch backends.

A good choice for optimizations with a moderate number of qubits and parameters with exact expectation values.

default.mixed

A mixed-state qubit simulator written in Python, with Autograd, JAX, TensorFlow, and Torch backends.

A good choice for simulating noisy circuits and quantum channels.

default.gaussian

A simple quantum photonic simulator written in Python.

A good choice for optimizing photonic systems.

lightning.qubit

A fast state-vector qubit simulator written with a C++ backend.

A good choice for optimizations with a moderate number of qubits and parameters, or when using stochastic expectation values.

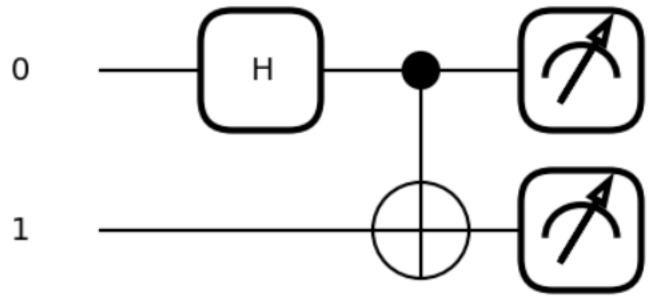
lightning.gpu

A fast state-vector qubit simulator utilizing the NVIDIA cuQuantum SDK for GPU accelerated circuit simulation.

A good choice for a large number of qubits and parameters, taking advantage of one or more GPUs for acceleration.

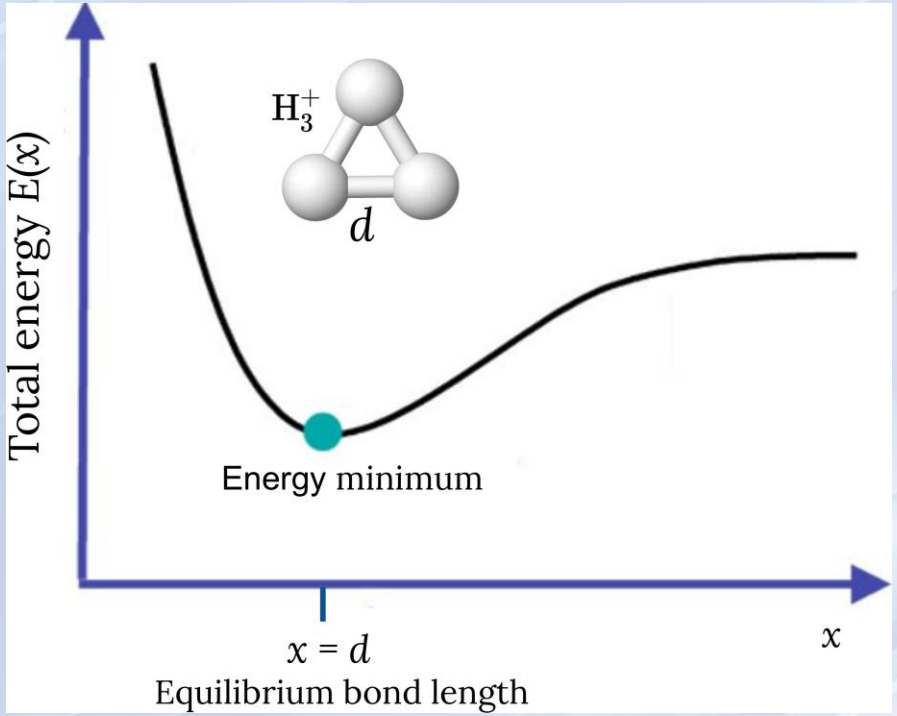
Pennylane: Get started

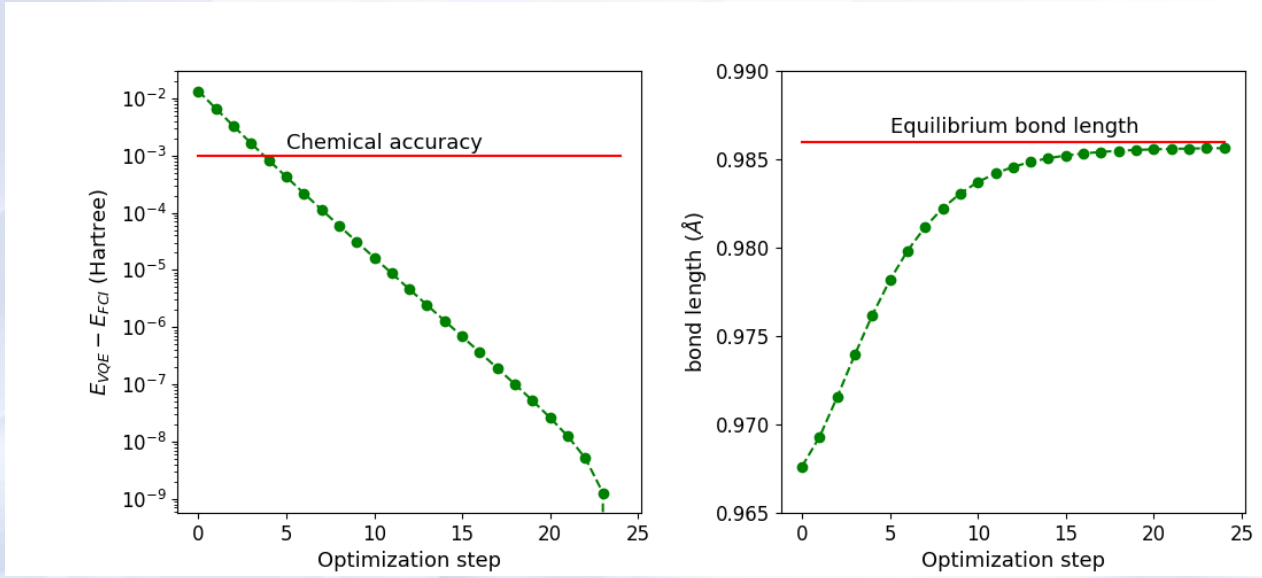
```python
# Import libraries
import pennylane as qml
from pennylane import numpy as np
# Create a device with 2 qubits
dev = qml.device('lightning.qubit', wires=2)
# Create a QNode with 2 entangled qubits
@qml.qnode(dev)
def circuit():
    qml.Hadamard(wires=0)
    qml.CNOT(wires=[0,1]) # [control qubit,target qubit]
    return qml.probs() # probability measurement. Output [P(00),P(),P(),P(11)]
# Run the circuit
circuit()
# Draw the circuit
qml.draw_mpl(circuit)()
```



Pennylane: Optimization of molecular geometries

Example of how the scope of variational quantum algorithms can be extended to perform quantum simulations of molecules involving both the electronic and the nuclear degrees of freedom

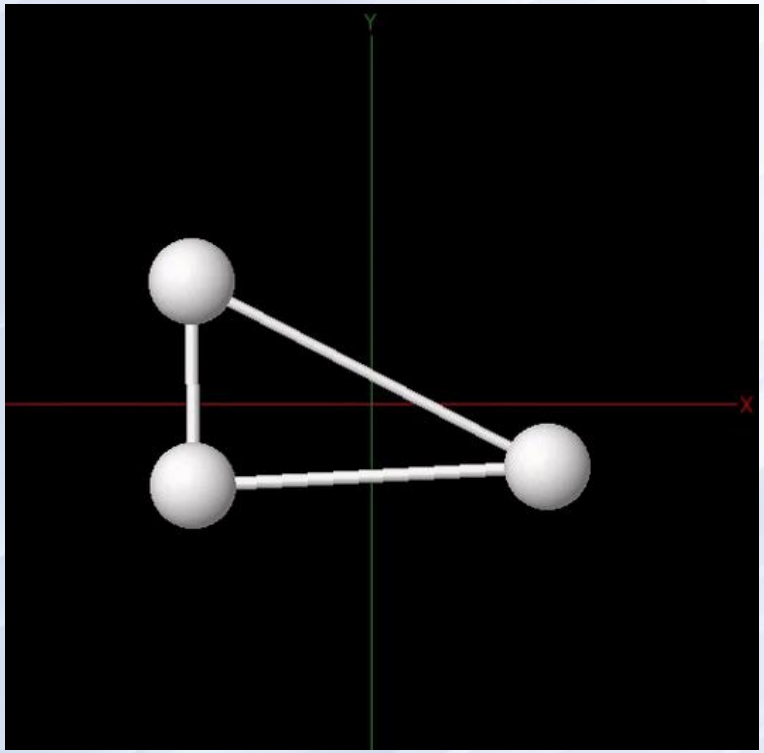




Tutorial here: https://pennylane.ai/qml/demos/tutorial_mol_geo_opt/

CINECA QUANTUM COMPUTING LAB

Pennylane: Optimization of molecular geometries



Snapshots' animation of the structure of the trihydrogen cation as the quantum algorithm was searching for the atomic equilibrium geometry.

Tutorial here: https://pennylane.ai/qml/demos/tutorial_mol_geo_opt/

CINECA QUANTUM COMPUTING LAB

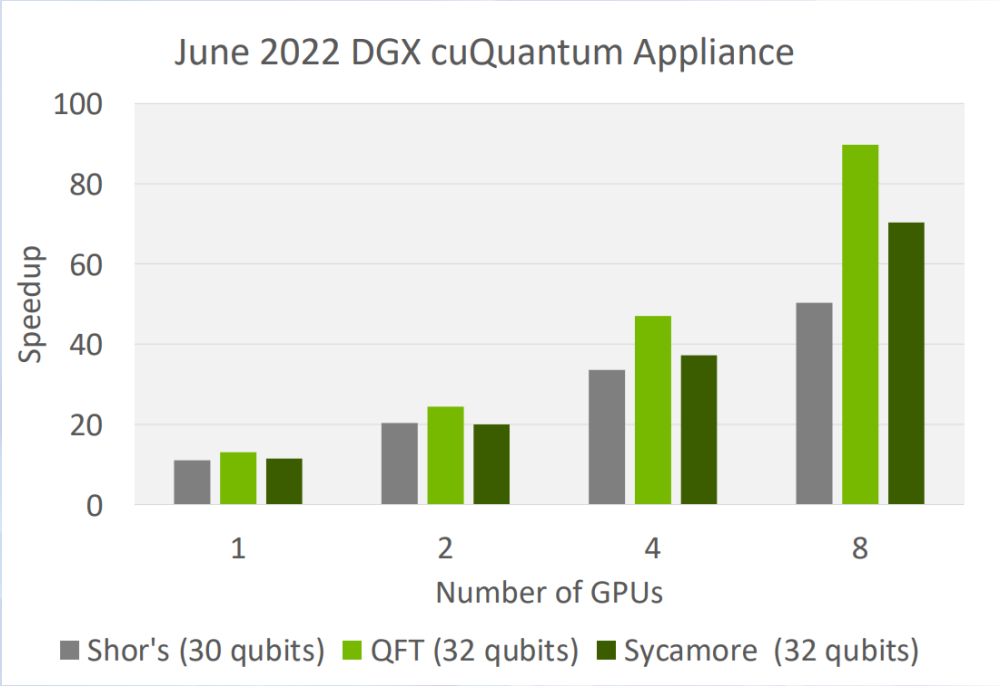
Hands-on

- **Exercise 1**

  Compare QFT (quantum fourier transform) solved with CirQ, PennyLane, Qiskit

  Hint: play with parameters (num. of qubits, with and without GPUs, num. Of GPUs, num.threads, ntasks-per-node, –cpus-per-task etc...)

Ideal result (CirQ):



- **Exercise 2** make same comparisons with QAOA and/or quantum volume algorithm