# Hands-on session based on Atos myQLM framework

Simone Perriello
Email simone.perriello@polimi.it

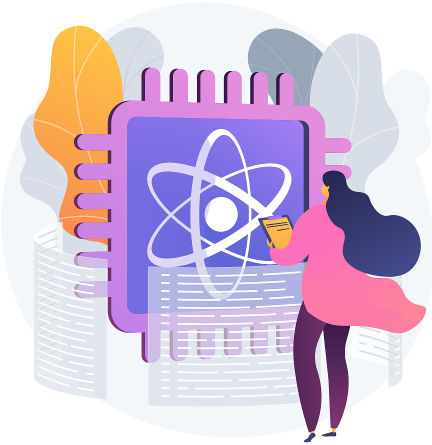
Dipartimento di Elettronica, Informazione e Bioingegneria - DEIB
Politecnico di Milano

March 24, 2022

# Plan

# Atos framework overview

- 2 main projects
  - myQLM: open-source, can run on any computer and OS
  - QLM: closed-source, run on dedicated supercomputer, available for academia (included PoliMi) and enterprises
- Different simulators are available, based both on historical and state-of-the-art proposals
- We will focus on simple simulators
  - gate-model representation of quantum operators
  - based on linear-algebra matrix operations

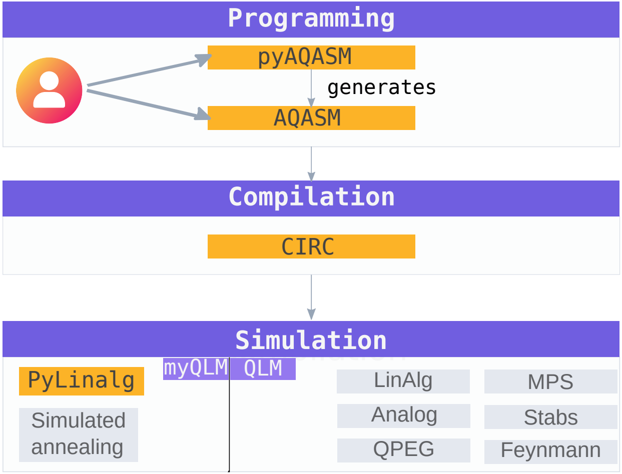


# Atos framework overview



- Main limiting factor in linear algebra simulation is memory
  - Given $n$ qubits, naive way to represent a state is by using a vector of $2^n$ complex numbers
- 2 linear algebra simulators developed by Atos
  - **PyLinalg** on *myQLM*, written in *Python* (with *numpy* libraries), it allows to simulate 20-25 qubits on standard laptops
  - **LinAlg** on *QLM*, closed-source, allows to simulate up to 41 qubits (using 60TiB of memory)

# Programming framework overview



# Before continuing



## Open myQLM notebooks

It will take time and we will need it later

https://github.com/Polimi-Courses/myqlm-notebooks/tree/polimi2022

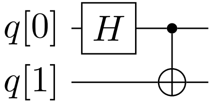
# Plan

# Atos Quantum Assembly (AQASM)



- Assembly language for quantum circuit description
  - no loop
  - no branching
  - no subroutines
- Standard gates defined
- Custom gates can be added

```
BEGIN
qubits 2
H q[0]
CNOT q[0], q[1]
END
```



# Gates and operators



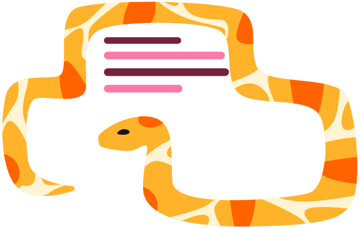
| Gate | Keyword | Qubits |
|---|---|---|
| Hadamard | H | 1 |
| Pauli X | X | 1 |
| Pauli Y | Y | 1 |
| Pauli Z | Z | 1 |
| Identity | I | 1 |
| Phase | S | 1 |
| $\pi/8$ | T | 1 |
| X rotation | RX[theta] | 1 |
| Y rotation | RY[theta] | 1 |
| Z rotation | RZ[theta] | 1 |

| Gate | Keyword | Qubits |
|---|---|---|
| CNOT | CNOT | 2 |
| CZ | CZ | 2 |
| iSWAP | ISWAP | 2 |
| $\sqrt{\text{SWAP}}$ | SQRTSWAP | 2 |
| Toffoli | CCNOT | 3 |

| Operator | Keyword |
|---|---|
| Conjugate | CONJ |
| Transpose | TRANS |
| Dagger | DAG |
| Control | CTRL |

# PyAQASM

- Python library for generating AQASM files
- Used to simplify generation of quantum circuits
  - Loops for repeating gates
  - Functions and subroutines
  - ...
- Allows hybrid programming model
  - Controls, subroutines, ... handled with *classical programming paradigm*
  - Generated quantum circuits implement *quantum paradigm*



# PyAQASM Gates and operators



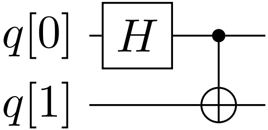
- Same gate set of AQASM
  - `H`, `X`, `CNOT`, `CCNOT`
- Operators on gates become functions

| Operator | AQASM | PyAQASM |
|---|---|---|
| Conjugate | CONJ | `conj()` |
| Transpose | TRANS | `trans()` |
| Dagger | DAG | `dag()` |
| Control | CTRL | `ctrl(nbctrls=1)` |

# Example I — EPR pair



```python
from qat.lang.AQASM import *

pr = Program()
qr = pr.qalloc(2)
pr.apply(H, qr[0])
pr.apply(CNOT, qr[0], qr[1])
# equivalently, pr.apply(CNOT, qr)
```



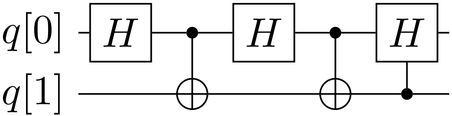
# Hands on — EPR pair



```
from qat.lang.AQASM import *

pr = Program()
qr = pr.qalloc(2)
pr.apply(H, qr[0])
pr.apply(CNOT, qr[0], qr[1])

    # We can export our
    # program into an AQASM file
    pr.export('pr.aqasm')
```

# Example II

```
from qat.lang.AQASM import *

pr = Program()
qr = pr.qalloc(2)
for _ in range(2):
    pr.apply(H, qr[0])
    pr.apply(CNOT, qr[0], qr[1])
pr.apply(H.ctrl(1), qr[1], qr[0])
```



# CIRC object

- Binary format of a quantum circuit
- Generated through compilation of AQASM code
  - either via command-line utility or directly through PyAQASM
- Pivot of all QLM/myQLM stack
  - simulators
  - optimizers and plugins (not seen here)
  - more trivially, functions for circuit display

# Hands on — EPR pair circuit



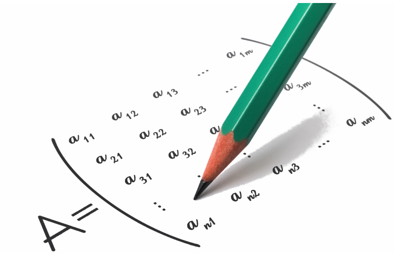
```
from qat.lang.AQASM import *

pr = Program()
qr = pr.qalloc(2)
pr.apply(H, qr[0])
pr.apply(CNOT, qr[0], qr[1])
```

```
# We can export our program into a circuit object
cr = pr.to_circ()
# and save it to a file
cr.dump('pr.circ')
```

# PyLinalg



- Based on linear algebra
  - $n$ qubits represented by a $2^n$ vector
  - memory is the bottleneck
- Simulation time function of number of gates
- PyLinalg exploits numpy libraries
- Different simulation modes available for the same circuit
  - generate full state vector
  - generate state vector of a subset of qubits
  - strictly emulate a QPU and generate a single basis state



# Hands on — EPR pair simulation



```python
from qat.lang.AQASM import *

pr = Program()
qr = pr.qalloc(2)
pr.apply(H, qr[0])
pr.apply(CNOT, qr[0], qr[1])
```

```python
from qat.qpus import PyLinalg
qpu = PyLinalg()
# generate a job containing the circuit
# and some other information
job = cr.to_job()
# Result contains all the states with non-zero amplitude
result = qpu.submit(job)
```

# Plan

# PyAQASM advanced

- Classical operations
  - Hands on — Teleportation circuit
- Abstract and parametrized gates
  - Hands on — CCNOT decomposition in Clifford + T gate set
  - Hands on — Deutsch-Jozsa algorithm with abstract oracles
- Quantum subroutines and linking
  - Hands on — Deutsch-Jozsa algorithm with real oracles
  - Hands on — Bernstein-Vazirani algorithm
- A useful application for the Quantum Fourier Transform

# Plan

# My quantum experiments



- All the algorithms seen here can be found on my github repository `https://github.com/tigerjack/qat-experiments`
- A huge collection of useful extension to the quantum languages (routines, qubit management, . . . ) used in my research projects can be found here `https://github.com/tigerjack/qat-utils`

# Atos Resources



- https://atos.net/en/lp/myqlm
- https://join.slack.com/t/myqlmworkspace/shared_invite/zt-nvtt5hk3-BX53Dg5YhZaYWRnRoDtLUA?
- https://atos.net/en/solutions/quantum-learning-machine

# Thanks for your attention Email

simone.perriello@polimi.it