ESILV

myQLM

Gaëtan Rubez, PhD
Senior expert in Quantum Computing
2022

Content overview

01. myQLM

Atos

Expand your Quantum Programmer Community with myQLM

Scientists: You are currently using the Atos QLM and you want to prepare your code and run them on your laptop?

Students: You want to start programming Quantum algorithms using the same framework as your professors?

Tech enthusiasts: You want to discover Quantum programming using an accessible and user-friendly environment?

➔ https://atos.net/en/lp/myqlm

Atos

myQLM

WRITE
your own quantum algorithms

Discover AQASM and pyAQASM

Explore Jupyter Notebook tutorials
Adapt QLIB algorithms

RUN & TEST
your quantum circuits

On your laptop using pyLinalg or your own simulator

On your Atos QLM

SHARE
tips and codes with the community

Create myQLM user communities

Collaborate with other frameworks' users

Atos

What functionalities are included in myQLM?

PROGRAMMING

AQASM
Assembly language to build quantum circuits

pyAQASM
Python extension to AQASM

CIRC
Binary format of quantum circuits

QLIB
AQASM & pyAQASM libraries
QRAM, oracle emulator, arithmetic libraries and more!

INTEROP
Connectors' source codes: build your own!

Open Source

SIMULATION

Open Source

SIMULATOR
pyLinalg
Source code of this simulator: build your own!

Atos

pyAQASM

Overview pyAQASM

Python

AQASM

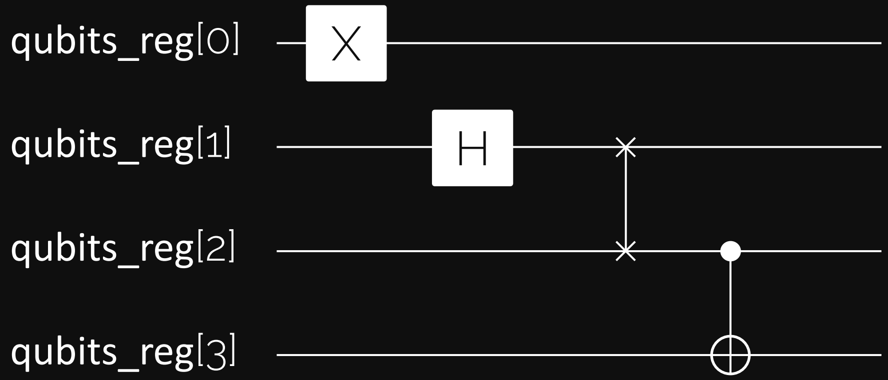
Ease the developpement

Writing circuits in pyAQASM

Import functions
Create your program
Allocate registers of qubits
Apply gates
Create your circuit
Display your circuit

```
from qat.lang.AQASM import Program, X, H, CNOT, SWAP
#Create a Program
my_program = Program()
#Allocate some qubits
qubits_reg = my_program.qalloc(4)
#Apply some quantum Gates
my_program.apply(X, qubits_reg[0])
my_program.apply(H, qubits_reg[1])
my_program.apply(SWAP, qubits_reg[1], qubits_reg[2])
my_program.apply(CNOT, qubits_reg[2], qubits_reg[3])
#Export this program into a quantum circuit
my_circuit = my_program.to_circ()
#And display it!
%qatdisplay my_circuit
```

Writing circuits in pyAQASM




```
from qat.lang.AQASM import Program, X, H, CNOT, SWAP
#Create a Program
my_program = Program()
#Allocate some qubits
qubits_reg = my_program.qalloc(4)
#Apply some quantum Gates
my_program.apply(X, qubits_reg[0])
my_program.apply(H, qubits_reg[1])
my_program.apply(SWAP, qubits_reg[1], qubits_reg[2])
my_program.apply(CNOT, qubits_reg[2], qubits_reg[3])
#Export this program into a quantum circuit
my_circuit = my_program.to_circ()
#And display it!
%qatdisplay my_circuit
```

List of constant available gates

Gate name	Keyword	Arity
Hadamard	H	1
Pauli X	X	1
Pauli Y	Y	1
Pauli Z	Z	1
Identity	I	1
S gate	S	1
T gate	T	1

Gate name	Keyword	Arity
Controlled NOT	CNOT	2
SWAP	SWAP	2
iSWAP	iSWAP	2
$\sqrt{SWAP}$	SQRTSWAP	2
Toffoli	CCNOT	3

Atos

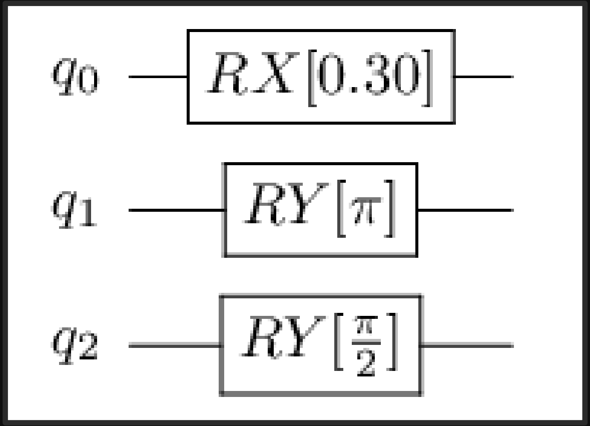
pyAQASM – Parametrized gates

Gate name	Keyword	Arity
Rotation x-axis	RX[θ]	1
Rotation y-axis	RY[θ]	1
Rotation z-axis	RZ[θ]	1
Phase shift	PH[θ]	1

pyAQASM – Parametrized gates

```
from qat.lang.AQASM import RX, RY
import math

#Apply some quantum Gates
my_program.apply(RX(0.3), qubits_reg[0])
my_program.apply(RY(math.pi), qubits_reg[1])
my_program.apply(RY(math.pi/2), qubits_reg[2])
```



Operations on gates

Operation name	Keyword	Example	Note
control	ctrl	G.ctrl()	The first qubit of the list is the controller
dagger	dag	G.dag()	Creates the dagger of the gate G
transpose	trans	G.trans()	Creates the transpose of the gate G
conjugate	conj	G.conj()	Creates the conjugate of the gate G

Simulating circuits in pyAQASM

Import functions
Get a simulator
Create your job
Submit your job
Print the result

```
#import one Quantum Processor Unit Factory
from qat.qpus import PyLinalg
#Create a Quantum Processor Unit
qpu = PyLinalg()
#Create a job
job = my_circuit.to_job()
#Submit the job to the QPU
result = qpu.submit(job)
#Iterate over the final state vector to get all final components
sample in result:
    print("State %s amplitude %s" % (sample.state, sample.amplitude))
```

Simulating circuits in pyAQASM

```
State |1000> amplitude
(0.7071067811865475+0j)
State |1011> amplitude
(0.7071067811865475+0j)
```

```
#import one Quantum Processor Unit Factory
from qat.qpus import PyLinalg
#Create a Quantum Processor Unit
qpu = PyLinalg()
#Create a job
job = my_circuit.to_job()
#Submit the job to the QPU
result = qpu.submit(job)
#Iterate over the final state vector to get all final
components
sample in result:
   print("State %s amplitude %s" % (sample.state,
sample.amplitude))
```

pyAQASM: Job

```
job = circuit.to_job(*options*) #creating job from circuit.
results = qpu.submit(job)       #submitting job to QPU instance, getting results.
```

Circuit execution modes:

- Full distribution (*default case*)
- Strictly emulate
- Directly compute observable averages *(Advance topic)*

Atos

pyAQASM: Job

- Full distribution (*default case*):

```
job = circuit.to_job()              #creating job from circuit to get full distribution.
results = qpu.submit(job)           #submitting job to QPU instance, getting results.
```

results contains **all states with non-zero amplitude**.

The job was created with *default arguments*:

for example **nbshots = 0**, by convention, it means the **qpu** returns the **best it can do**.

pyAQASM: Job

- Strictly emulate:

```
job = circuit.to_job(nbshots = 6, aggregate_data=False)#creating job from circuit to get 6 measures.
results = qpu.submit(job)                              #submitting job to QPU instance, getting results.
```

results contains **6 measurements** that you can print:

```
for state in results:
    print(state)
```

```
Sample(state=|00>, probability=None, amplitude=None, intermediate_measurements=None, err=None)
Sample(state=|00>, probability=None, amplitude=None, intermediate_measurements=None, err=None)
Sample(state=|11>, probability=None, amplitude=None, intermediate_measurements=None, err=None)
Sample(state=|11>, probability=None, amplitude=None, intermediate_measurements=None, err=None)
Sample(state=|00>, probability=None, amplitude=None, intermediate_measurements=None, err=None)
Sample(state=|11>, probability=None, amplitude=None, intermediate_measurements=None, err=None)
```

pyAQASM: Job

- Strictly emulate (with *aggregate_data*):

```
job = circuit.to_job(nbshots = 6)        #creating job from circuit to aggregate 6 measures.
results = qpu.submit(job)                #submitting job to QPU instance, getting results.
```

results contains **one unique sample** per possible output with an **empirical estimation of the probability**:

```
for state in results:
      print(state)
```

```
Sample(state=|00>, probability=0.5, amplitude=None, intermediate_measurements=None, err=0.16666666666666666)
Sample(state=|11>, probability=0.5, amplitude=None, intermediate_measurements=None, err=0.16666666666666666)
```

Atos

pyAQASM: Job

- Subset of qubits:

```
job = circuit.to_job(qubits=[0])         #creating job from circuit only on the first qubit
results = qpu.submit(job)                #submitting job to QPU instance, getting results.
```

results contains **all possible states with a non-zero probability** for the subset of qubits.

```
for state in results:
    print(state)
```

```
Sample(state=|0>, probability=0.4999999999999999, amplitude=None, intermediate_measurements=None, err=None)
Sample(state=|1>, probability=0.4999999999999999, amplitude=None, intermediate_measurements=None, err=None)
```

pyAQASM: help

```
help(circ.to_job)
```

```
Help on method to_job in module qat.core.wrappers.circuit:

to_job(job_type='SAMPLE', qubits=None, nbshots=0, aggregate_data=True, amp_threshold=9.094947017729282e-13,
**kwargs) method of qat.core.wrappers.circuit.Circuit instance
    Generates a Job containing the circuit and some post
    processing information.

    Args:
        job_type (str): possible values are "SAMPLE" for computational basis sampling
            of some qubits, or "OBS" for observable evaluation (see :py:mod:`qat.application.observables`
            for more information about this mode).
        qubits (optional, list<int>, list<QRegister>): the list of qubits to measure (in "SAMPLE" mode).
            If some quantum register is passed instead, all the qubits of the register will be…
```

Thank you!

For more information please contact:
gaetan.rubez@atos.net