Quantum Programming Demo

Ivano Pullano
Global product manager - quantum computing
25/05/2022

Programming workflow in Atos QLM

Gate-based quantum computing

Very general workflow in gate-based computing

- Encode the problem input into the qubit register
- Encode the algorithm into a set of quantum gates
- Perform the calculation
- Readout the results for further processing

Standard workflow in Atos QLM

- Use `qat.lang.AQASM` to build a `Program` object including quantum/classical registers and gates
- Use `Program`'s `to_circ()` method to build a `Circuit` object including the final gate implementation
- Use `Circuit`'s `to_job()` method to build a Job object including the execution parameters
- Build a `QPU` object using Atos' quantum emulators, third-party QPUs, or plugins
- Build the `Result` object with the QPU's `submit(job)` method
- Iterate on the `Result` object to extract the needed parameters for further operation

Atos

Example of code

A basic Bell pair

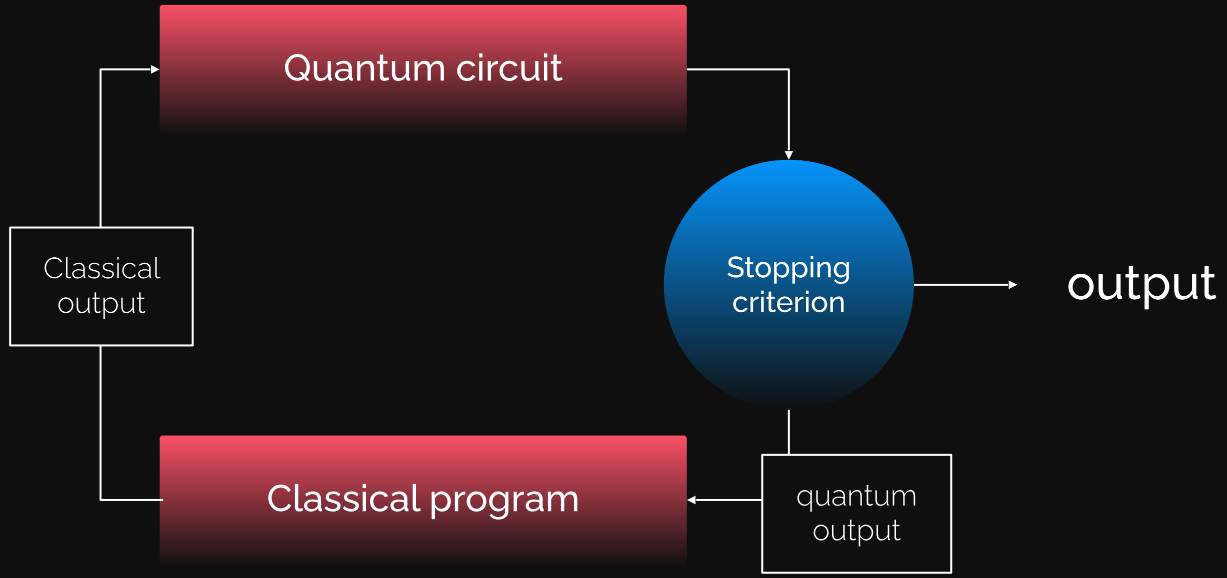
Code

```python
from qat.lang.AQASM import Program, H, CNOT
from qat.qpus import get_default_qpu

# Create a circuit
qprog = Program()
qbits = qprog.qalloc(2)
H(qbits[0])
CNOT(qbits[0], qbits[1])
circuit = qprog.to_circ()

# Create a job
job = circuit.to_job(nbshots=100)

# Execute
result = get_default_qpu().submit(job)
for sample in result:
    print("State %s: probability %s +/- %s" % (sample.state, sample.probability, sample.err))
```

Output

```
State |00>: probability 0.53 +/- 0.050161355804659184
State |11>: probability 0.47 +/- 0.050161355804659184
```

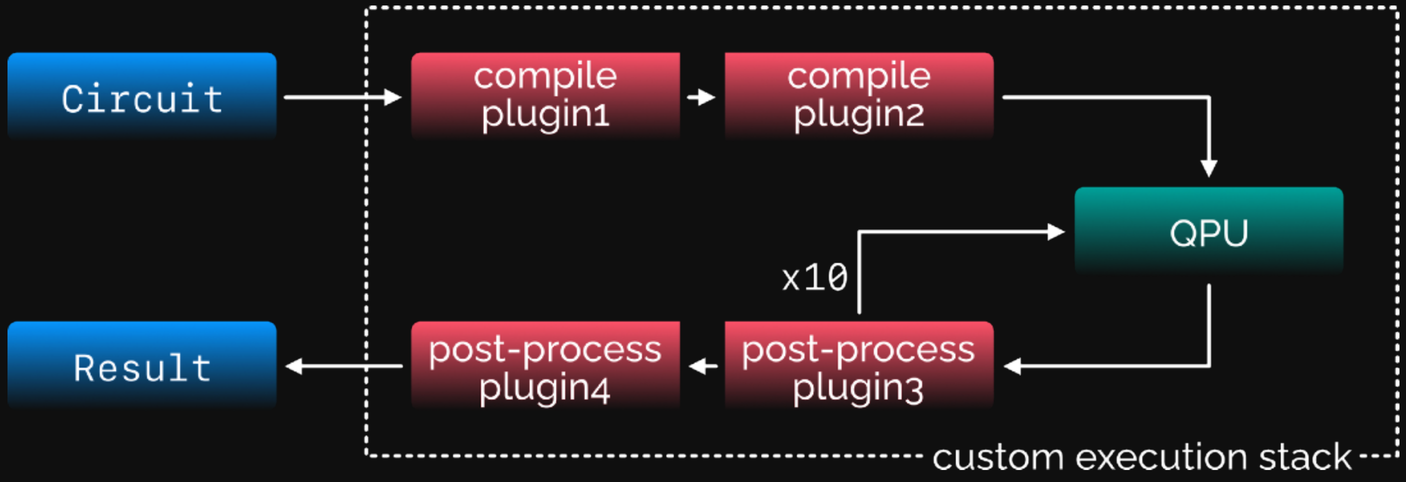
Quantum Approximate Optimisation Algorithm



Atos

Custom execution stacks

Extend the capabilities of any computing backend



Extension of any QPU object to a stack

```
my_stack = plugin1 | plugin2 | plugin3 | plugin4 | my_qpu
```

How to model very large problems with good fidelity

Atos quantum emulators

Linear Algebra emulator (LinAlg)

- Encode the state of the QPU into a state vector
- Since each qubit can interact with any other, you must consider all the possible combinations
- N qubits → 2^N combinations, each one with a certain amplitude
- Encode each gate as a matrix that modifies the state vector
- The result is the vector that comes out of the various gates
- Huge RAM consumption

Matrix Product State emulator (MPS)

- Encode the state of the QPU into a sequence of single qubit states multiplied by a matrix that encodes the interactions with the others
- Weak qubit interactions → small matrices → simple calculations
- Very good when qubits either do not interact each other, or they only interact with their neighbours

Atos

How to model very large problems with good fidelity

Atos quantum emulators

QPEG emulator

- Developed as a joint effort between Atos and CEA
- Group up qubits into a network of “tiny" QPUs
- Treat each QPU using LinAlg
- Treat the interaction between groups using MPS
- Apply gates in layers
- Moving the resource consumption from RAM to CPU
- Double-digit precision on the results

Further resources

- myQLM documentation
  https://qlm.bull.com/doc/qat-tutorial-myqlm/index.html
- Atos QLM documentation
  https://qlm.bull.com/doc/qat-tutorial-qlm/index.html
- Quantum annealing benchmarking routines
  https://qlm.bull.com/doc/qat-tutorial-qlm/advanced_combinatorial_optimization.html
- myQLM GitHub page
  https://github.com/myQLM

Atos