Programmation quantique –Défis et opportunités

Cyril ALLOUCHE, directeur R&D Quantum Computing

6 Dec 2018

Summary

- Principles and applications of Quantum Programming
- Quantum hardware limitations
- The NISQ challenge and the need for numerical simulation
- Atos QLM

Atos

1

Quantum Programming and Applications

A Huge Paradigm Shift

Classic	Quantum
deterministic	probabilistic
create, overwrite, delete data	reversible computing: information can’t be destroyed
move and copy data	information can’t be cloned
massive readouts	as few readouts as possible
control flow + boolean operations	superposition + entanglement

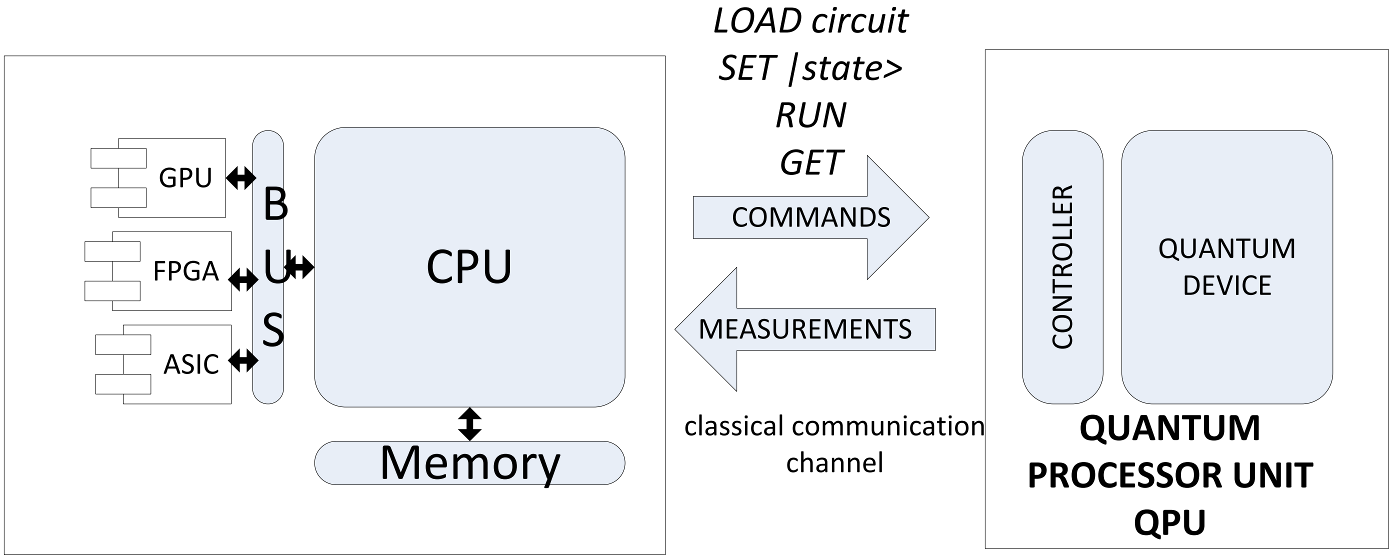
Theorem : No classical algorithm can be implemented on a quantum computer with a quantum speedup

Corolary: There will never exist an automatic converter classical –> quantum

Hybrid Computing

- Quantum computing will remain **specialized**
- Control flow do remain on classical computers
- Quantum Processors Units (QPU) as hardware accelerators
- Interface classical/quantum can only exchange classical information (ie not in superposition)
- Creating quantum information (superposition) from classical has an **exponential** cost

Programming Model



MAIN APPLICATION – MANAGE CONTROL FLOW AND DATA FLOW

QUANTUM SERVICE – DELEGATED, ONLINE, COMPUTATION

```python
from AQASM import *

def angleCum(mResultVec, k):
    theta = 0.0;
    for j in range(2, (k+1)):
        if (mResultVec[k - j] == 1):
            theta += 1.0 / (2.0**j)
    return 2.0*math.pi*theta

def powMod(a, x, N):
    res = a
    if (x == 0):
        return (a % N);
    for i in range(1, x):
        res = ((res*res) % N)
    return res

def ModInverse(a, N):
    k = 0
    p = 0
    q = 0
    while (q != 1):
        p = a * k
        q = p % N
        k = k + 1
    return (k-1)

def QFT(AQASM_prog, qbList):
    nbqubits = len(qbList)
    for i in range(0, nbqubits):
        AQASM_prog.apply(H, qbList[nbqubits - i - 1])
        for j in reversed(range(i+1, nbqubits)):
            phaseFac = 2.0*math.pi / (2** (j - i + 1))
            AQASM_prog.apply(PH(phaseFac).ctrl(), [qbList[nbqubits - i - 1]
```

Hybrid program

CPU

QPU

Quantum

Processor Unit

```python
from AQASM import *

def angleCum(mResultVec, k):
    theta = 0.0;
    for j in range(2, (k+1)):
        if (mResultVec[k - j] == 1):
            theta += 1.0 / (2.0**j)
    return 2.0*math.pi*theta

def powMod(a, x, N):
    res = a
    if (x == 0):
        return (a % N);
    for i in range(1, x):
        res = ((res*res) % N)
    return res

def ModInverse(a, N):
    k = 0
    p = 0
    q = 0
    while (q != 1):
        p = a * k
        q = p % N
        k = k + 1
    return (k-1)

def QFT(AQASM_prog, qbList):
    nbqubits = len(qbList)
    for i in range(0, nbqubits):
        AQASM_prog.apply(H, qbList[nbqubits - i - 1])
        for j in reversed(range(i+1, nbqubits)):
            phaseFac = 2.0*math.pi / (2** (j - i + 1))
            AQASM_prog.apply(PH(phaseFac).ctrl(), [qbList[nbqubits - i - 1]
```

Hybrid program

Executes classical instructions

CPU

QPU

Quantum

Processor Unit

```python
from AQASM import *

def angleCum(mResultVec, k):
    theta = 0.0;
    for j in range(2, (k+1)):
        if (mResultVec[k - j] == 1):
            theta += 1.0 / (2.0**j)
    return 2.0*math.pi*theta

def powMod(a, x, N):
    res = a
    if (x == 0):
        return (a % N);
    for i in range(1, x):
        res = ((res*res) % N)
    return res

def ModInverse(a, N):
    k = 0
    p = 0
    q = 0
    while (q != 1):
        p = a * k
        q = p % N
        k = k + 1
    return (k-1)

def QFT(AQASM_prog, qbList):
    nbqubits = len(qbList)
    for i in range(0, nbqubits):
        AQASM_prog.apply(H, qbList[nbqubits - i - 1]
        for j in reversed(range(i+1, nbqubits)):
            phaseFac = 2.0*math.pi / (2** (j - i + 1
```

```
BEGIN
qubits 12
H q[3]
H q[4]
H q[5]
H q[6]
H q[7]
H q[8]
H q[9]
H q[10]
H q[11]
CNOT q[2],q[0]
S q[2]
S q[1]
CNOT q[2],q[1]
H q[1]
CCNOT q[0],q[1],q[2]
S q[1]
CCNOT q[0],q[1],q[2]
T q[2]
CNOT q[0],q[2]
T q[0]
H q[1]
CCNOT q[2],q[1],q[0]
CCNOT q[1],q[0],q[2]
S q[2]
CCNOT q[0],q[2],q[1]
CNOT q[0],q[1]
T q[2]
CCNOT q[0],q[1],q[2]
T q[2]
CTRL(T) q[3],q[0]
CTRL(H) q[3],q[0]
```

Hybrid program

CPU

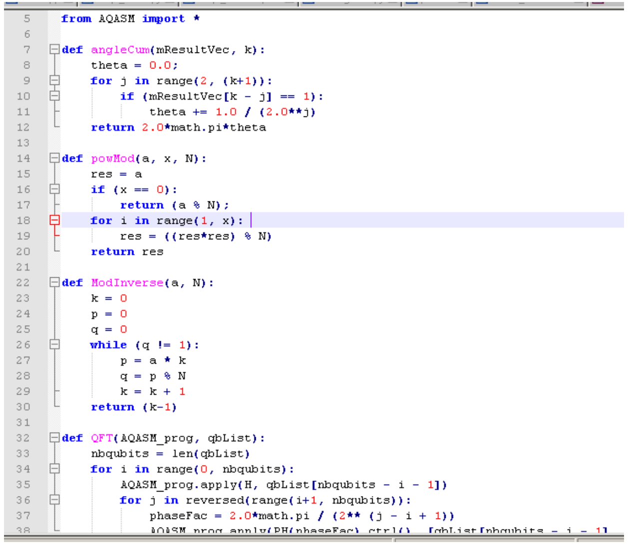
Delegate Quantum AQASM instructions to QPU

QPU

Quantum

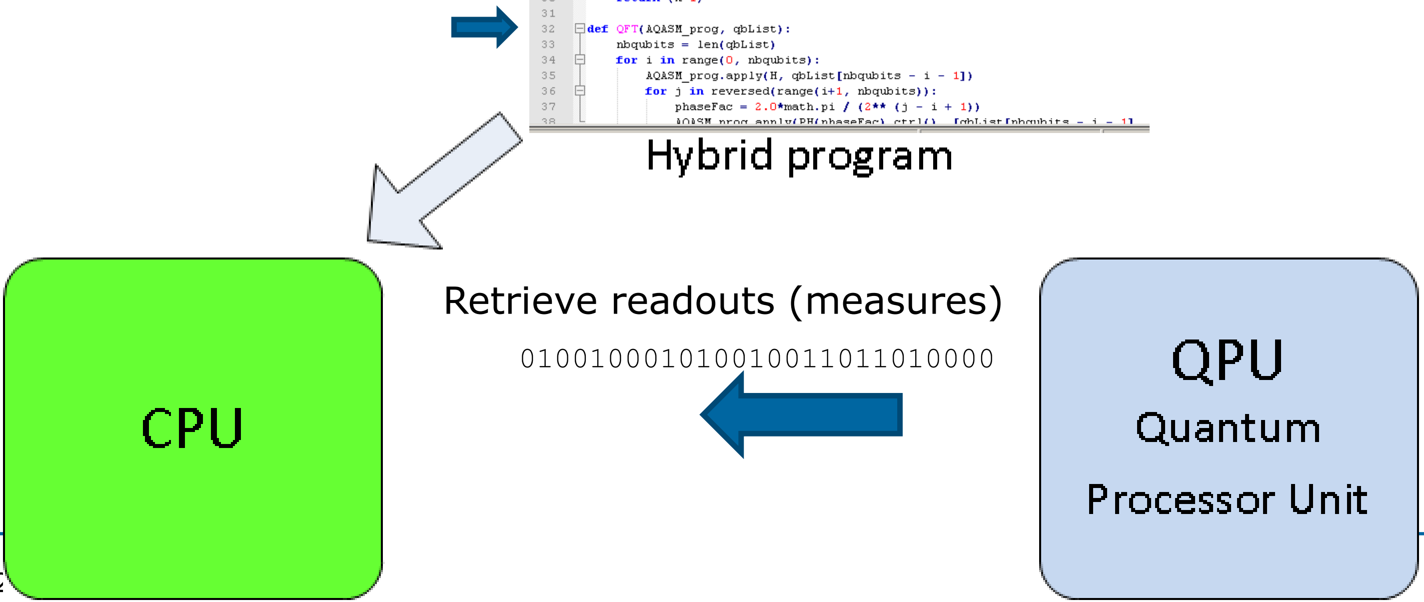
Processor Unit

Atos



```python
from AQASM import *

def angleCum(mResultVec, k):
    theta = 0.0;
    for j in range(2, (k+1)):
        if (mResultVec[k - j] == 1):
            theta += 1.0 / (2.0**j)
    return 2.0*math.pi*theta

def powMod(a, x, N):
    res = a
    if (x == 0):
        return (a % N);
    for i in range(1, x):
        res = ((res*res) % N)
    return res

def ModInverse(a, N):
    k = 0
    p = 0
    q = 0
    while (q != 1):
        p = a * k
        q = p % N
        k = k + 1
    return (k-1)

def QFT(AQASM_prog, qbList):
    nbqubits = len(qbList)
    for i in range(0, nbqubits):
        AQASM_prog.apply(H, qbList[nbqubits - i - 1])
        for j in reversed(range(i+1, nbqubits)):
            phaseFac = 2.0*math.pi / (2** (j - i + 1))
```

Hybrid program



```python
from AQASM import *

def angleCum(mResultVec, k):
    theta = 0.0;
    for j in range(2, (k+1)):
        if (mResultVec[k - j] == 1):
            theta += 1.0 / (2.0**j)
    return 2.0*math.pi*theta

def powMod(a, x, N):
    res = a
    if (x == 0):
        return (a % N);
    for i in range(1, x):
        res = ((res*res) % N)
    return res

def ModInverse(a, N):
    k = 0
    p = 0
    q = 0
    while (q != 1):
        p = a * k
        q = p % N
        k = k + 1
    return (k-1)

def QFT(AQASM_prog, qbList):
    nbqubits = len(qbList)
    for i in range(0, nbqubits):
        AQASM_prog.apply(H, qbList[nbqubits - i - 1])
        for j in reversed(range(i+1, nbqubits)):
            phaseFac = 2.0*math.pi / (2** (j - i + 1))
```

Hybrid program

Continue processing

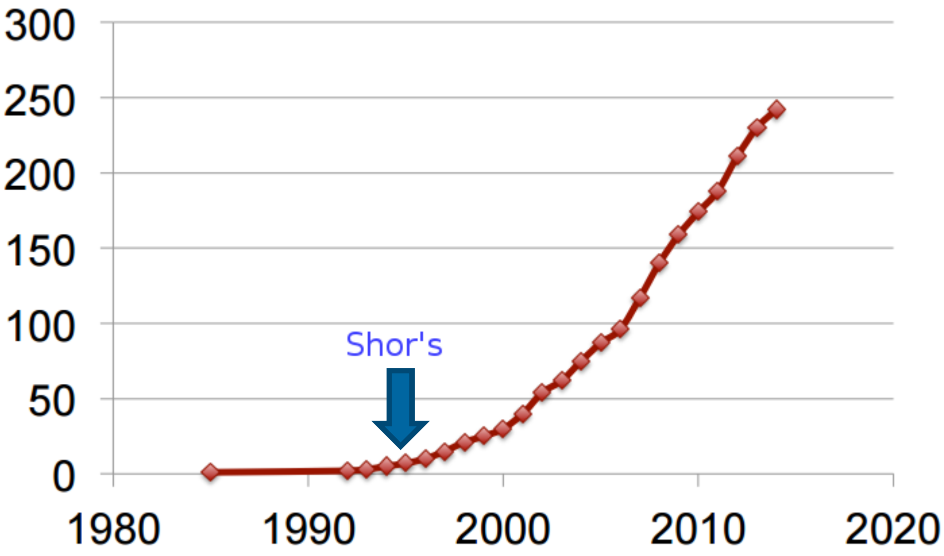
CPU

QPU

Quantum

Processor Unit

State of the Art of Quantum Algorithms



http://math.nist.gov/quantum/zoo/

Algorithms of Interest to HPC

- eigen computation
- linear system solving
- PDE
- combinatorial optimization
- Hamiltonian evolution
- .. and all those not yet invented

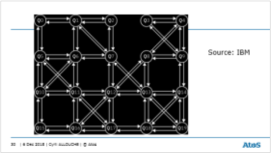
2 Quantum Hardware Limitations

Quantum Noise

- Noise is inherent to quantum physics
- So far, no efficient quantum error correction code has been invented
  - best known technique (« Surface Code ») costs a 30,000 overhead!
- Entanglement makes scaling **exponentially** hard

Other Technological Limitations

- ► Connectivity



- ► Operations fidelity (in particular measurement)
- ► Manufacturing bias
- ► Operational conditions
- ► …

3

The NISQ challenge and the need for numerical simulation

Noisy Intermediate Scale Quantum

- Algorithms have to deal with noise
- Limited resources : 50-1000 qubits
- Limited connectivity
- **strong game changer!**

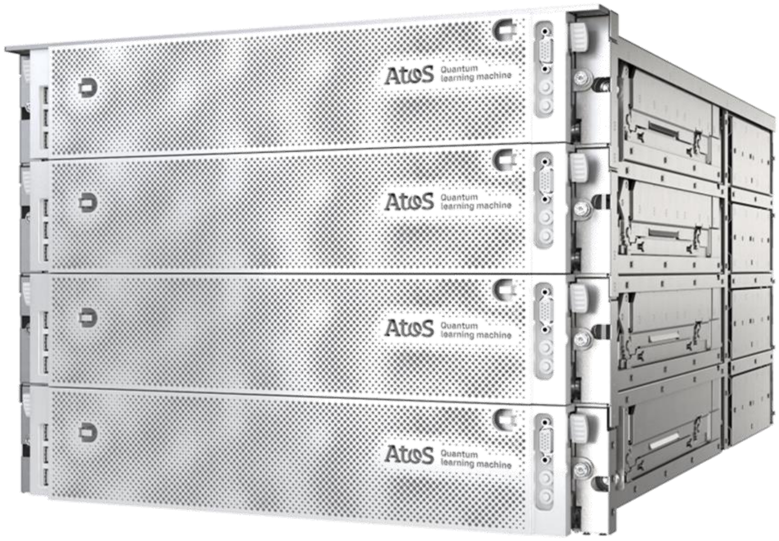
Numerical Simulation to Design NISQ Algorithms

- Exact simulation
- Hardware modeling and numerical simulation of noise
- Computation intensive optimization:
  - of resources
  - of noise tolerance

Atos

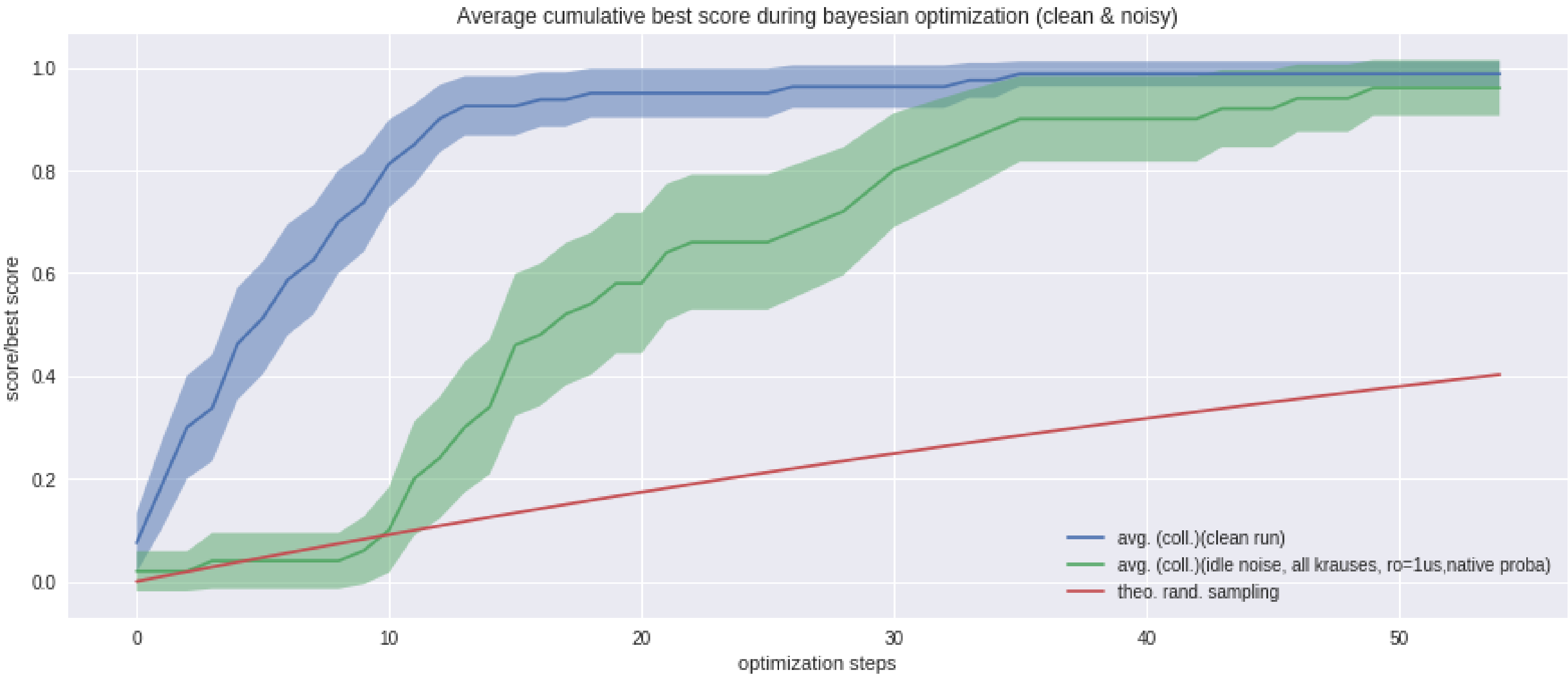
4 The Atos Quantum Learning Machine

Atos Quantum Learning Machine

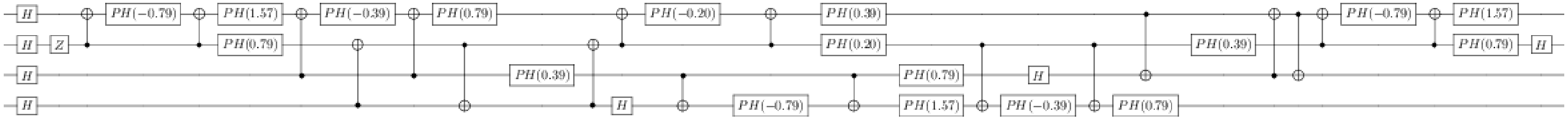


- Complete platform for research, training & experimentation of quantum software
- HW and SW developed in-house by Atos
- Python based quantum/hybrid computing
- Universal – can be connected to a real QPU
- Simulation capability of **41** fully entangled qubits
- **Optimisation of quantum code for quantum hardwares**
- **Numerical simulation of quantum noise in qubits**
- Fully extensible – users can develop their own language, compiler, simulator…

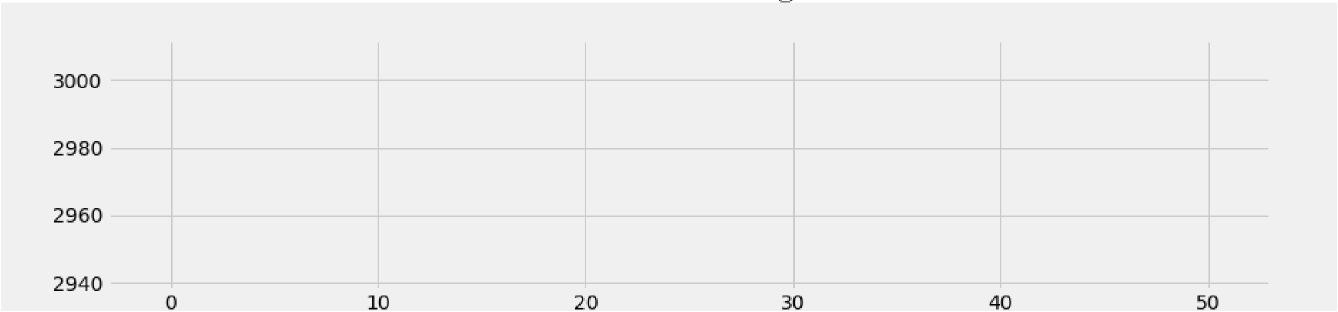
Working a NISQ Algorithm – the QAOA Algorithm



Optimizing an Algorithm



Minimize overall idling time



Atos QLM Customers

Oak Ridge National Laboratory

DTU Technical University of Denmark

Jülich Forschungszentrum

SURF SARA

CCRT Centre de calcul recherche et technologie

Université de Reims Champagne-Ardenne

Argonne National Laboratory

Total

FH Oberösterreich University of Applied Sciences Upper Austria

Conclusion

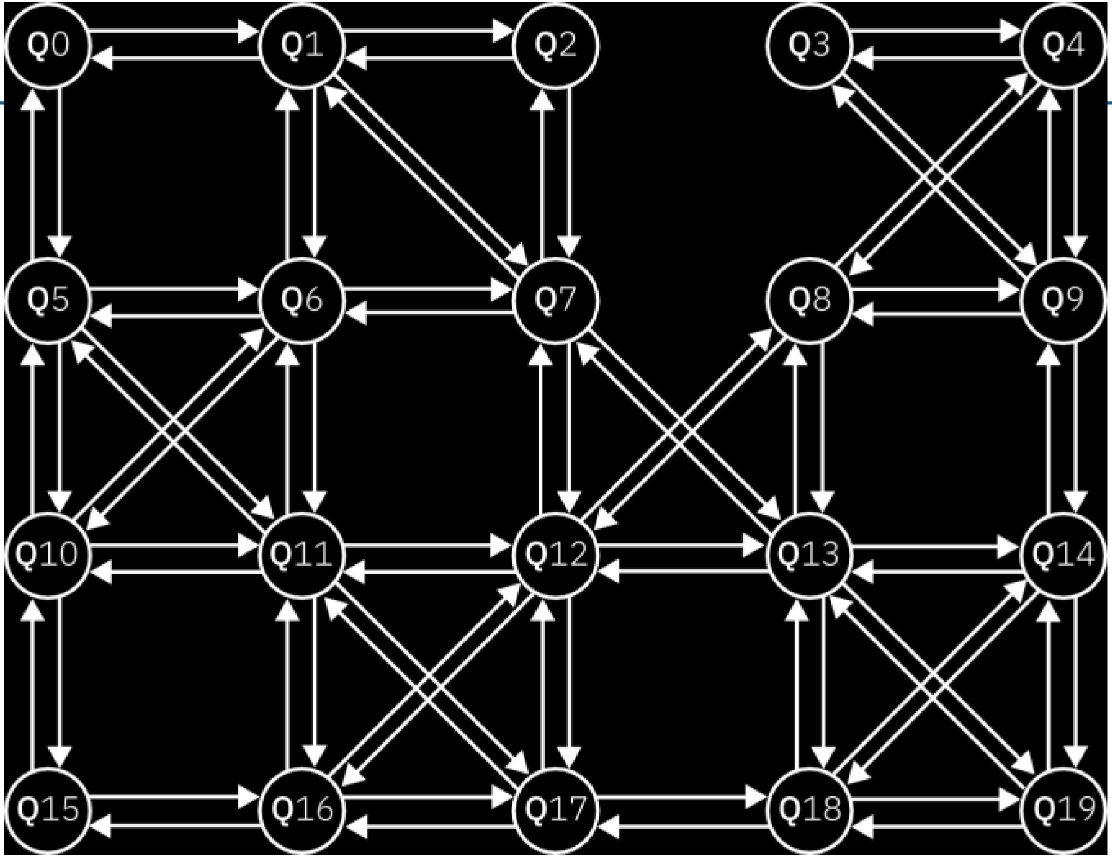
- Quantum Programming is a new paradigm
- Hardware is (re)shaping the game
- New Algorithmics has to be invented
- QC will not replace classical computers – coprocessor based hybrid architectures
- Numerical simulation is an indispensable tool

Thanks

For more information please contact:

Cyril.Allouche@atos.net

Appendices


Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
Q9
Q10
Q11
Q12
Q13
Q14
Q15
Q16
Q17
Q18
Q19

Source: IBM