QC WARE

Q2B

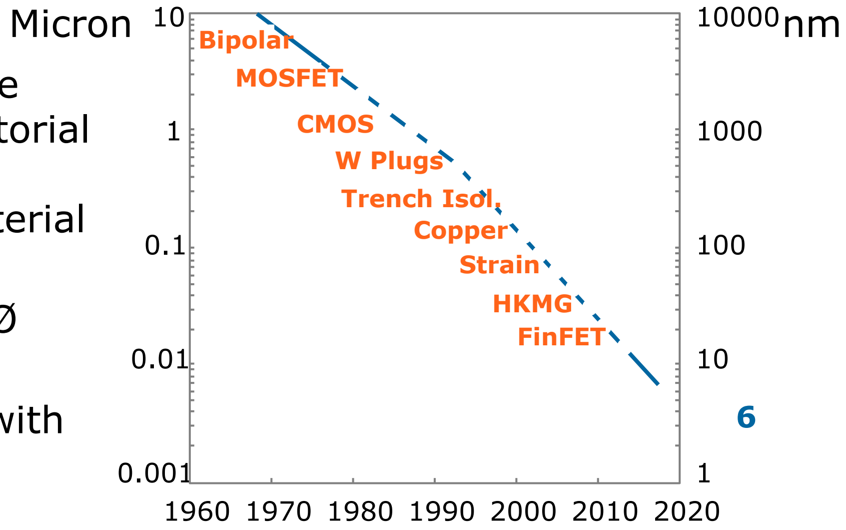
2017

Atos Quantum Learning Machine: today and in the future...

Philippe Duluc, CTO, Big Data & Security
Scott Hamilton, Senior Expert Hardware and Firmware

The two Quantum disruptions for Atos

- Computing
  - technological progress requests more and more compute power (big data revolution, combinatorial explosion, data analytics, artificial intelligence, complex simulation, long-term simulation, material science, molecular chemistry, etc.)
  - Moore law declining, Silicium atom : 0,11 nm Ø today fabrication process < 10 nm
  - quantum information brings speed up to deal with combinatorial explosion



- Cybersecurity
  - Shor has shown the way of exponential speedup, breaking virtually most of asymmetric cryptographic algorithms that are securing Internet (RSA, DH, ECDH, etc.)
  - also Grover against symmetric cryptographic algorithms
  - need to develop quantum-safe asymmetric algorithm

Atos' Vision

- We're at the early beginning of the 2nd Quantum Revolution

- A **new kind** of computing power to emerge, and a **new kind** of algorithms (probabilistic against deterministic, no cloning)

- The challenge is not just building physical qubits:
  - Quantum software : very new and largely unexplored area
  - Quantum-powered computing architectures : hybrid systems to be designed from scratch with industrial requirements (reliability, fidelity, maintainability…)
  - interface between quantum hardware and software : a new industrial challenge
  - optimization is vital in early stages
  - application, use-cases, adoption & dissemination

Atos Quantum : a long-term strategic R&D investment of disruptive innovation

- Position Atos as European industrial leader in Quantum Computing: 100,000 people, €12 billion, 72 countries, €300 million R&D, first European in computing and cyber-security

- High level Advisory Board, chaired by Atos CEO : Serge Haroche (Nobel prize), Cédric Villani (Field medalist), Alain Aspect, David Di Vincenzo, Artur Ekert, Daniel Esteve

- Atos Quantum R&D laboratory set up in 2016 near Paris

intel Bull atos technologies

Atos R&D Program – Four Pillars

1	**Quantum Programming Platform**	Complete programming and simulation environment for quantum software/hardware developers and for education/training
2	**Quantum Algorithms**	Atos' own research, focused on Quantum Machine Learning, one of the most promising application areas of QC
3	**Next Generation Architectures**	Designing the new quantum-powered accelerators for supercomputers or hybrid systems
4	**Quantum safe cryptography**	Preparing the cryptographies and hardware security modules, resistant to quantum computer attacks

The Quantum Learning Machine (QLM)

An appliance

Atos QLM features
Intel® Xeon® processors

Atos QLM Appliance

Atos QLM Software

Programming environment

QPU emulator
Optimizers
Simulators
Noise model

Atos QLM Hardware

Optimized In-memory Infrastructure
Scalable and Modular

What makes us different?

Atos QLM features
Intel® Xeon® processors

- First and unique commercial simulator (since July 2017)
- First customers (North-America, Europe), user group to come
- On premises (security, availability, performance, IP protection)
- Powerful simulation (40 Qubits, 24 Tbytes in-memory)
- Interoperable with other quantum workflows/systems
- For both researchers and education
- Hardware ready (test & run) and hardware agnostic, no lock in
- Optimization at compiler level and at simulator level
- Optimization of circuits with respect to physical noises (ECCM)
- Perfect tool for preparing and saving runs on existing quantum hardware

atos technologies

Programming and simulation : present targets

- Simulation of logical qubits (zero-noise)
  - **market** : quantum learners , algorithmic researchers, software developers
  - QLM based on in-memory dense servers (24 Tbytes) and patents (software architecture)
- Simulation of physical qubits (noise models)
  - **market** : quantum hardware designers, quantum algorithm implementers
  - partnerships with quantum HW labs and experimenters
  - HW agnostic (library as extensive as possible)
- Optimization of quantum software
  - **market** : quantum software developers
  - topological optimizer, quantum gates economizer, etc.
- Hamiltonian simulation
  - **market** : researchers in chemistry and material science
  - digital quantum simulation (extension of Feynman initial proposal)

Quantum computing : next targets

- QLM coupled to quantum demonstrator
  - **market** : HW dependent software developers, HW dependent algorithm implementers, quantum supremacy pioneers, quantum HW designers for calibration
  - demonstrator of real quantum HW (20-50 physical Qubits)
  - complete appliance QLM/demonstrator, or access to demonstrator via cloud
  - same software for simulation and for run, noise models improvement
  - HW agnostic (no exclusivity) : partnerships with HW providers
- quantum system
  - **market :** quantum-supremacy users (chemistry, material science, etc.)
  - real quantum system (40-100 logical Qubits) in one or more QPU (distributed QC)
  - real use-cases, coupled with traditional IT (FPGA, GPU, etc.)
  - QLM as programming interface, and for optimization and rapid testing on qubits subsets
  - choice of one HW technology, monitoring of others

Atos QLM functional scope

PROGRAMMING

AQASM
Assembly language to build quantum circuits

pyAQASM
Python extension to AQASM

CIRC
Binary format of quantum circuits

QLIB
AQASM & pyAQASM libraries

INTEROP
Connectors with other frameworks

EMULATION

QPU
Quantum processing unit emulation

OPTIMIZATION

RBO
Rule based optimizer

Circuit Optimizer
Generic circuit optimizer

NNIZER
Topology constraint solver

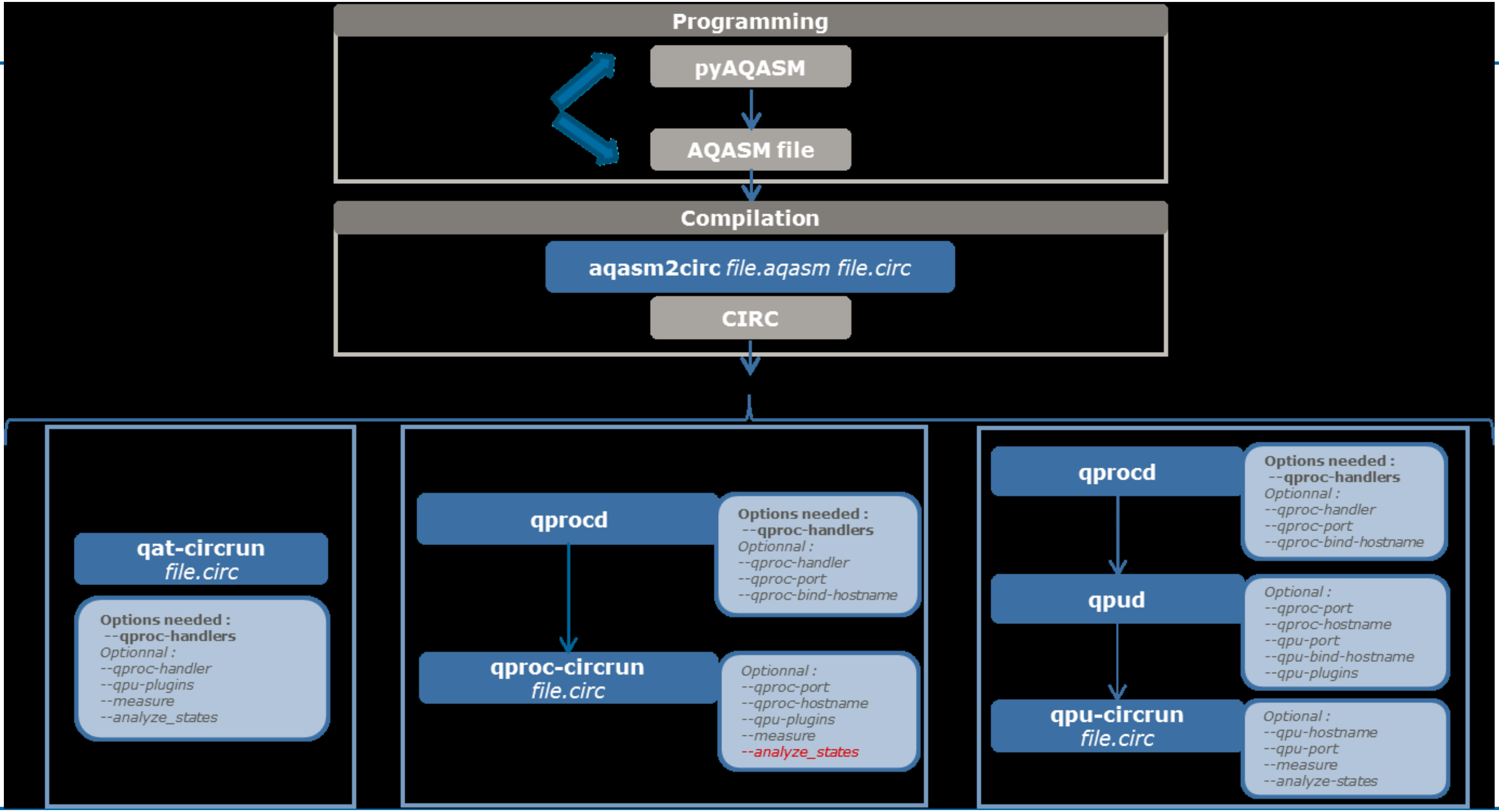
SIMULATION

SIMULATORS
Simulation modules

PHYSICS
Physical Noise models

SIM OPTIMIZER
Best Simulator dynamic selection

Standard Workflow



Quantum hybrid programming on QLM

- Hybrid programming paradigm
- Controls, subroutines and classical data manipulation is done on classical CPU
- Quantum code is offloaded to a QPU

```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 - j - 1], qbList[nbqubits - i - 1]])
```

Hybrid program

CPU

QPU
Quantum
Processor Unit

Quantum hybrid programming on QLM

```
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], qbList[nbqubits - j - 1]])
```

Hybrid program

Executes classical instructions

CPU

QPU
Quantum
Processor Unit

Quantum hybrid programming on QLM

```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], qbList[nbqubits - j - 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

Quantum hybrid programming on QLM

```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 - j - 1], qbList[nbqubits - i - 1]])
```

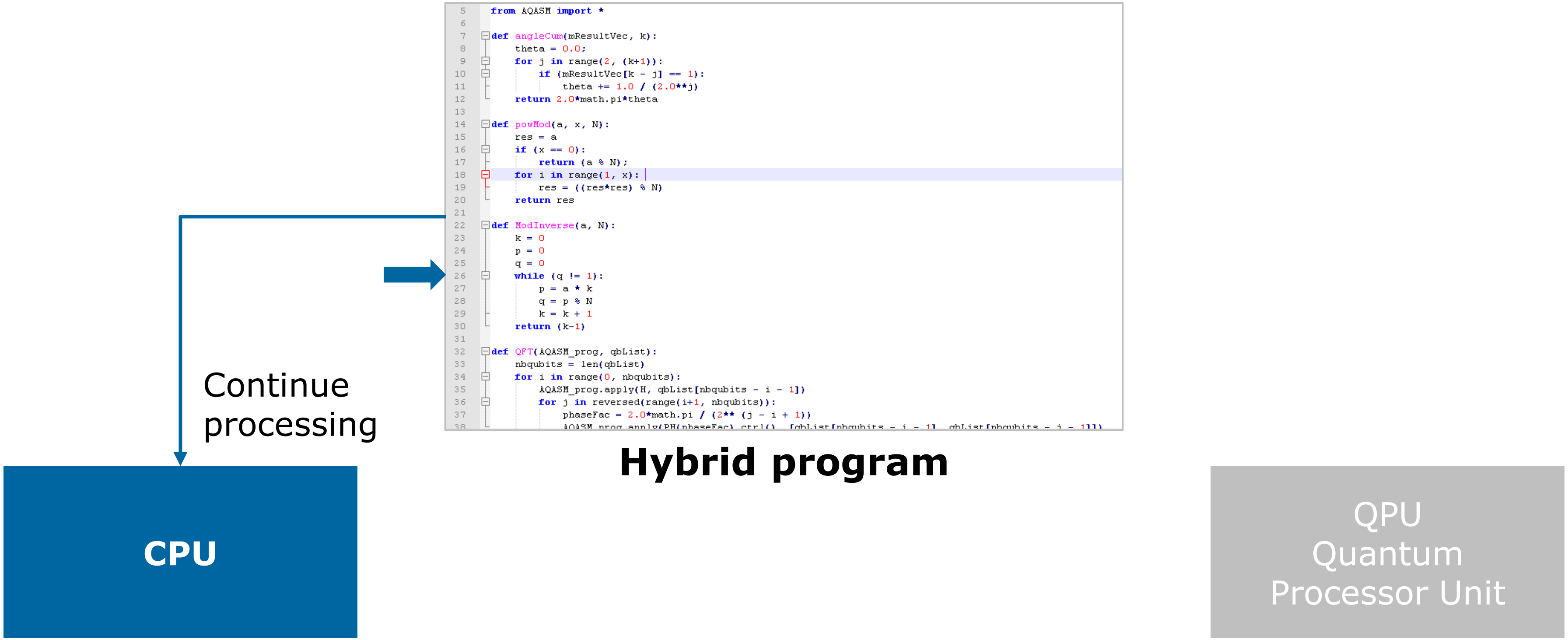
Hybrid program

CPU

QPU
Quantum
Processor Unit

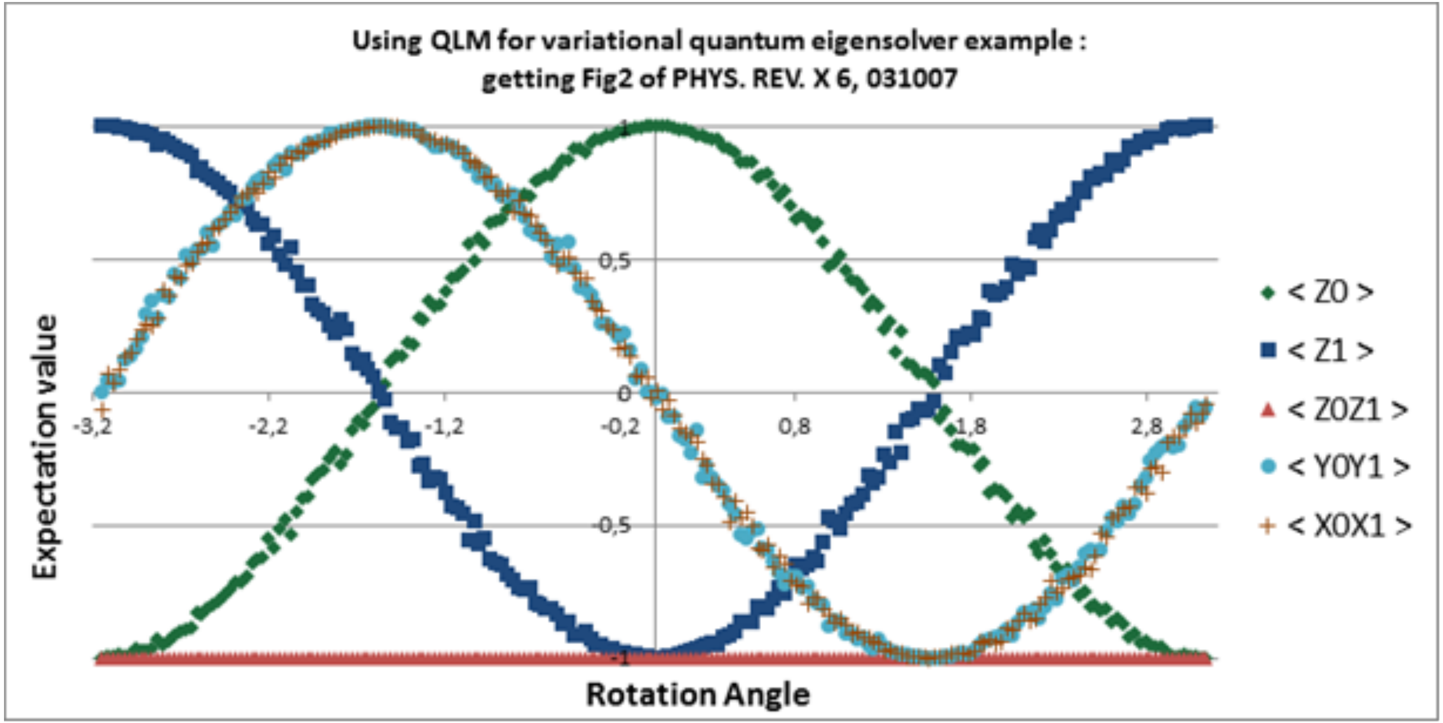
Retrieve readouts (measures)
01001000101001001101101000 0

Quantum hybrid programming on QLM




```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))
```

Example Eigensolver



H2 Electronic model courtesy of Alex McCaskey, Oak Ridge National Lab

Thanks

For more information please contact:

Philippe Duluc, CTO, Big Data & Security: philippe.duluc@atos.net
Scott Hamilton, Senior Expert Hardware and Firmware: scott.hamilton@atos.net

Bull
atos technologies

QC WARE

Q2B

2017