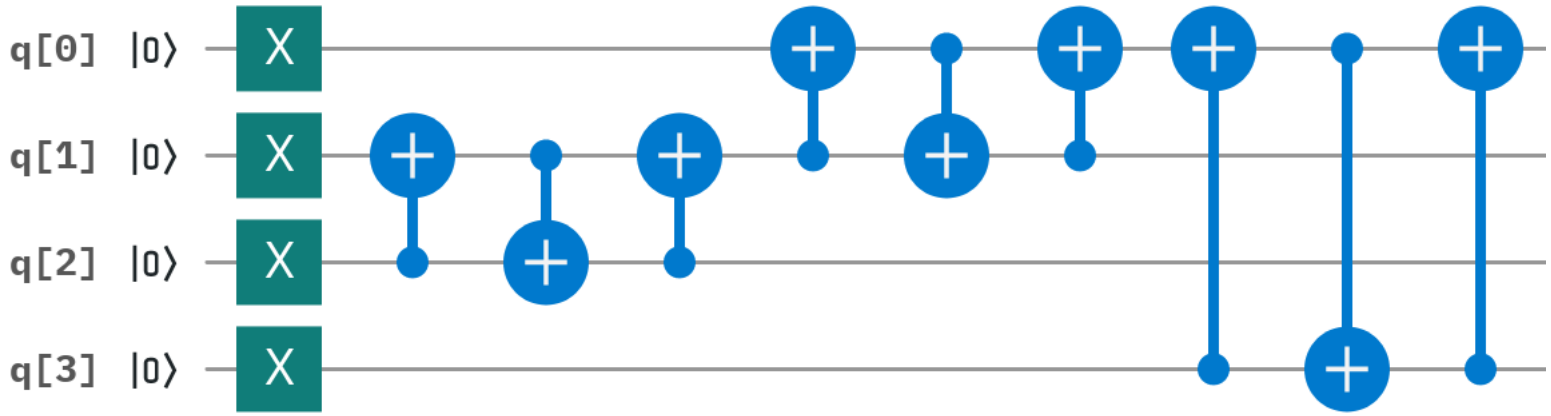

Quantum Computing Experiments

on Real Hardware and Simulators

Fanel Donea

CSIRO SCIENTIFIC COMPUTING, MODELLING AND DATA TEAM
www.csiro.au

Shor's Algorithm for Factorisation (P.W. Shor, 1994)

If N is the integer to be factorised, the algorithm proceeds as follows.

① Choose a random integer a, with $a < N$.

② Compute $g = gcd(a, N)$.

③ If $g \neq 1$, then g is a factor of N, so the (accidental) solution is $(g, N/g)$. END.

④ If $g = 1$ (very likely for large N), find the period r of the cyclic function $f(x) = a^x (mod\ N)$

⑤ If r is odd, RESTART.

⑥ If $a^{r/2} (mod\ N) = -1 (mod\ N)$, RESTART.

⑦ The solution is immediate: the factors are $gcd(a^{r/2} + 1, N)$ and $gcd(a^{r/2} - 1, N)$. END

Running the Algorithm 'by Hand' — A Lucky Choice

Modulo arithmetics trick: do NOT expand large powers, use previously calculated values and the property: $ab \,\%\, n = [(a \,\%\, n) \cdot (b \,\%\, n)] \,\%\, n$. We use the $\%$ notation here for brevity.

For $N = 15, a = 2$, $gcd(2, 15) = 1$ and we need the period of $f(x) = 2^x \,\%\, 15$.

$f(1) = 2^1 \,\%\, 15 = 2 \,\%\, 15 = 2$
$f(2) = 2^2 \,\%\, 15 = 4 \,\%\, 15 = 4$
$f(3) = 2^3 \,\%\, 15 = 8 \,\%\, 15 = 8$
$f(4) = 2^4 \,\%\, 15 = 16 \,\%\, 15 = 1$
$f(5) = 2^5 \,\%\, 15 = 32 \,\%\, 15 = 2$ ← We got 2 again after just 4 steps!

The period is $r = 4$ (even), $r/2 = 2$ and $a^{r/2} \,\%\, N = 2^2 \,\%\, 15 = 4$ is different than $-1 \,\%\, 15 = 14$. So the two factors are:
$gcd(4 + 1, 15) = gcd(5, 15) = 5$ and $gcd(4 - 1, 15) = gcd(3, 15) = 3$

$$15 = 3 \cdot 5 \quad !!!!$$

Running the Algorithm 'by Hand' — An Unlucky Choice

For $N = 7, a = 5$, $gcd(5, 7) = 1$ and we need to find the period of $f(x) = 5^x \,\%\, 7$. The calculations are harder this time.

$f(1) = 5^1 \,\%\, 7 = 5$
$f(2) = 5^2 \,\%\, 7 = 25 \,\%\, 7 = 4$
$f(3) = 5^3 \,\%\, 7 = (5 \cdot 5^2) \,\%\, 7 = (5 \cdot 4) \,\%\, 7 = 20 \,\%\, 7 = 6$

We started using the modulo trick mentioned before! We have to!!!

$f(4) = 5^4 \,\%\, 7 = 2$
$f(5) = 5^5 \,\%\, 7 = 3$
$f(6) = 5^6 \,\%\, 7 = 1$
$f(7) = 5^7 \,\%\, 7 = 5$ ← We got 5 again after 6 steps! Period is $r = 6$.

r is even and $r/2 = 3$. All good for now, but...

$a^{r/2} \,\%\, N = 5^3 \,\%\, 7 = 6$
$-1 \,\%\, N = -1 \,\%\, 7 = 6$

So we reached a dead end, we have to choose another 'random' value for a (in this case, no value will really help, we know that 7 is prime).

Remarks About the Algorithm

- The algorithm can run on both classical and quantum hardware. The classical version becomes useless very quickly.
- Amazingly, for something that threatens to bring down the Internet, it involves only elementary mathematics.
- Not all iterations from ① to ⑦ lead to a solution. There is an element of randomness even when the code is run on classical hardware.
- In case anyone noticed, step ⑦ is **not** recursive and not an NP problem! We do **not** have to collect common factors at the lowest powers etc etc. Euclid showed that 2400 years ago!!
- The period finding step ④ is the one that benefits from the power of quantum computing, mainly because of the non-classical way in which the Fourier transform can be implemented.
- Once NP is out of the way, $P + P = P$!!!

Qiskit Codes, Web Interface / Q Experience

- Qiskit — Python API. Easy to install (e.g. under Anaconda).
- RUN `shor_classical.py` (No Qiskit, nothing quantum here!)
- The implementation of Shor's quantum algorithm used here is from *Realization of a Scalable Shor Algorithm*, by T.Monz et al., arXiv:1507.08852v1, [quant-ph], 2015, further adapted by C. Corbett Moran, in *Mastering Quantum Computing*, Packt, 2019
- RUN `./set_backend.py`
- RUN `./launch_job-5q-REAL.py`
- RUN `./shor_quantum-SIMULATOR.py`
- RUN `./shor_quantum-REAL.py`
- Web based interface still has bugs, some limitations and frequent changes in the job submission API.
- Other notable API: QCEngine (JavaScript, by E. R. Johnston)
- Success=50%. $P + P = P$! It's ok to re-run the code! With 4000 qubits in one place...

Screenshot of a Previous Run — Shor Classical

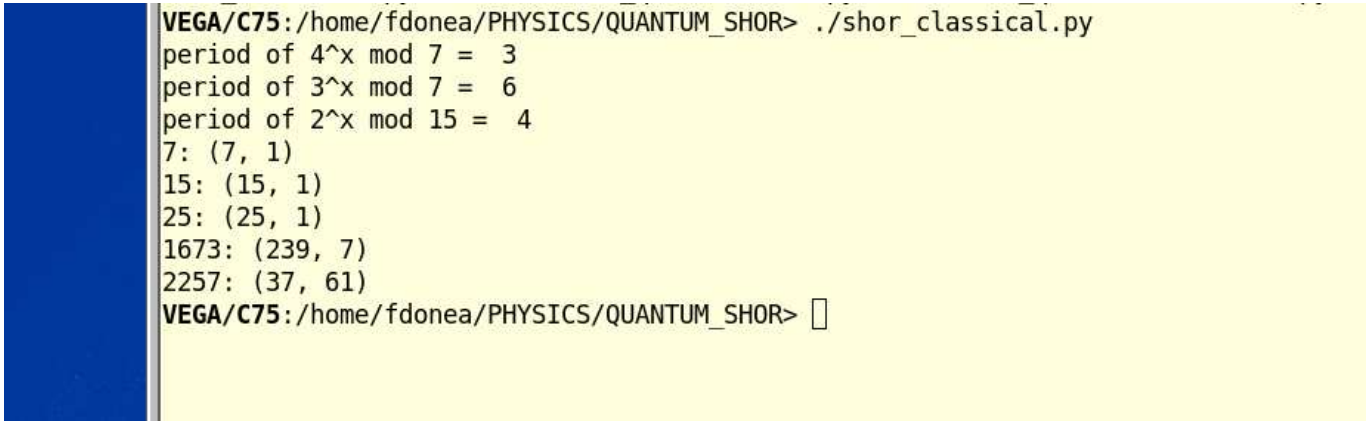

Figure 1: Output from classical version.

Screenshot of a Previous Run — Backends in Qiskit

```
VEGA/C75:/home/fdonea/PHYSICS/QUANTUM_SHOR> ./set_backend.py

--- Load account from disk. ---
[<AccountProvider for IBMQ(hub='ibm-q', group='open', project='main')>]

--- Select a provider and list its backends. ---
backends= [<IBMQSimulator('ibmq_qasm_simulator') from IBMQ(hub='ibm-q', group='open', project='ma
up='open', project='main')>, <IBMQBackend('ibmq_16_melbourne') from IBMQ(hub='ibm-q', group='open
(hub='ibm-q', group='open', project='main')>, <IBMQBackend('ibmq_ourense') from IBMQ(hub='ibm-q',

---- Selecting a backend by name ---
Selected backend =  ibmq_16_melbourne

--- Select backends that are real quantum dev, and operational: ---
[<IBMQBackend('ibmqx2') from IBMQ(hub='ibm-q', group='open', project='main')>, <IBMQBackend('ibmq_
ject='main')>, <IBMQBackend('ibmq_vigo') from IBMQ(hub='ibm-q', group='open', project='main')>, <
='open', project='main')>]

--- Select backends that are real q.dev, have>10 qubits + operational: ---
[<IBMQBackend('ibmq_16_melbourne') from IBMQ(hub='ibm-q', group='open', project='main')>]

--- Select the least busy 5 qubit device. ---
ibmq_vigo
VEGA/C75:/home/fdonea/PHYSICS/QUANTUM_SHOR>
```

Figure 2: Qiskit - Setting the backend.

Screenshot of a Previous Run — Launching a Job

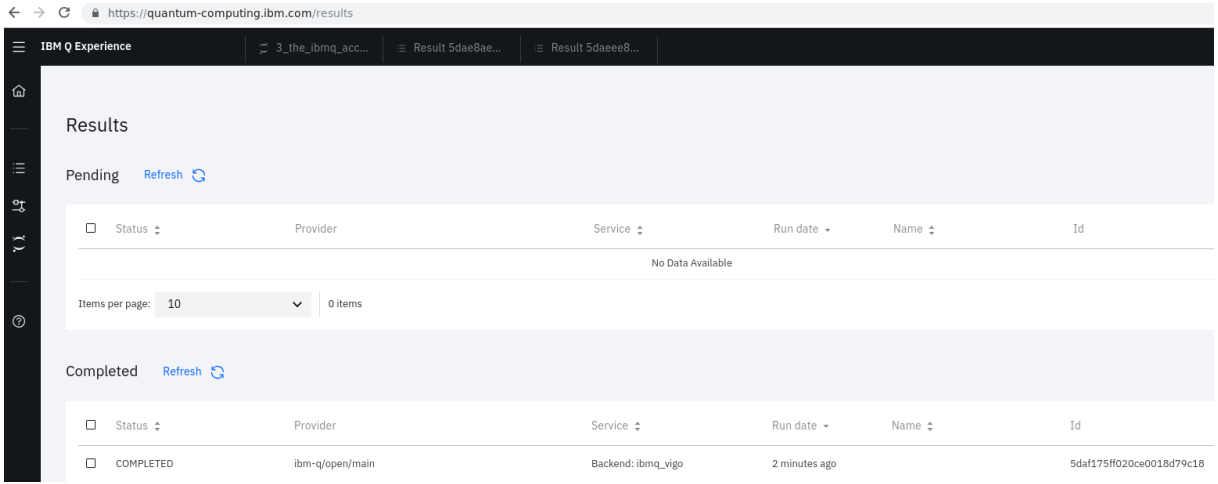

Figure 3: Qiskit - Launching a job programatically. Retrieving it in the web interface.

Screenshot of a Previous Run — Shor Quantum

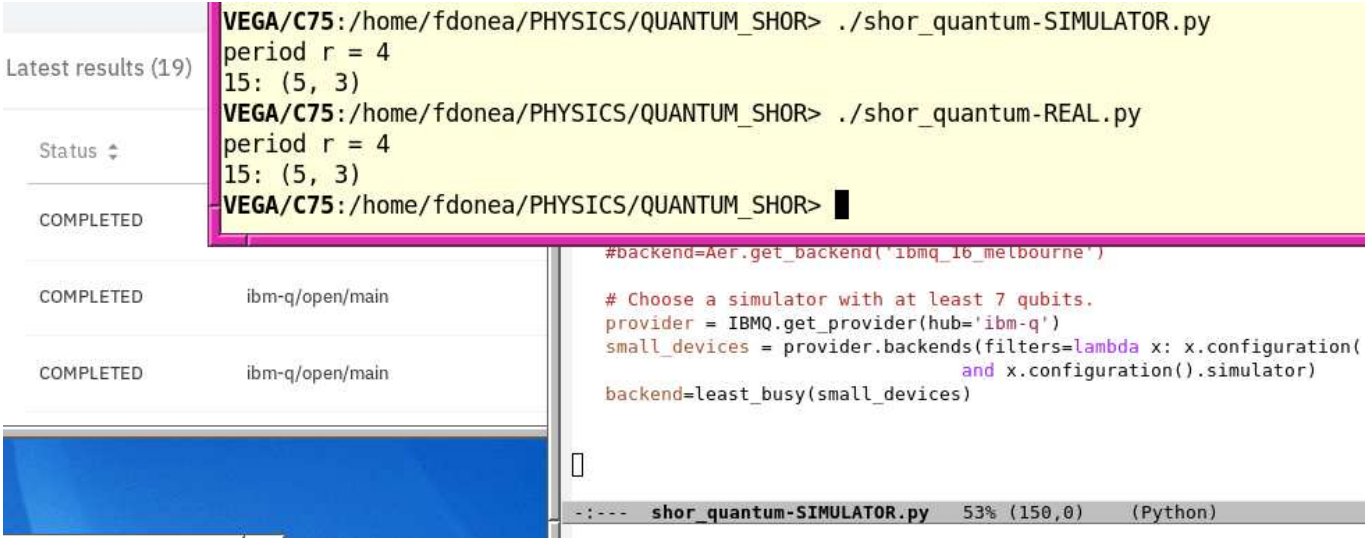

Figure 4: Output from simulator (`ibmq_qasm_simulator`), then from real device `ibmq_16_melbourne`

Future Plans etc

- Writing a simulator from scratch. Yes, adding to the existing hundreds (or thousands?).
- Developing an original algorithm at this point seems impossible, because most problems seem to have an algorithm waiting...
- ...Except one for which Google did not produce a link (won't tell which!).
- Good books (personal preferences):
  - *Quantum Computing for Computer Scientists* / Yanofsky and Mannucci, CUP, 2008 — The best introduction.
  - *Mastering Quantum Computing with IBM QX* / Corbett Moran, Packt, 2019. — Qiskit, very clear.
  - *Programming Quantum Computers* / Johnston et al., O'Reilly, 2019. — QCEngine
  - *Introduction to Quantum Mechanics, 3rd ed* / Griffiths and Schroeter, CUP, 2018 — good for learning QM.

Thank You

CSIRO Scientific Computing — Modelling and Data Team

Fanel Donea

t +61 3 9545 2188

e fanel.donea@csiro.au

w https://bitbucket.csiro.au/projects/QC