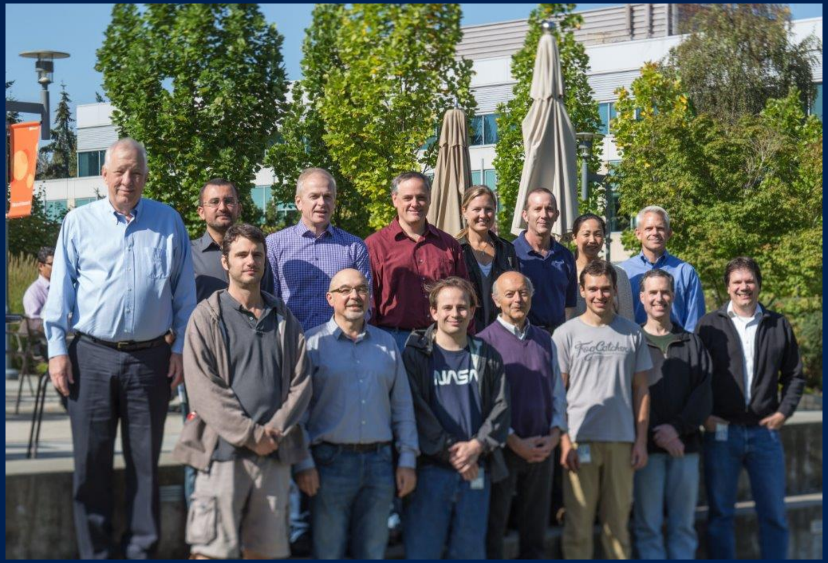
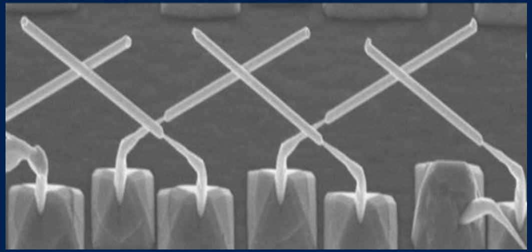
LIQ$Ui|\rangle$ tutorial

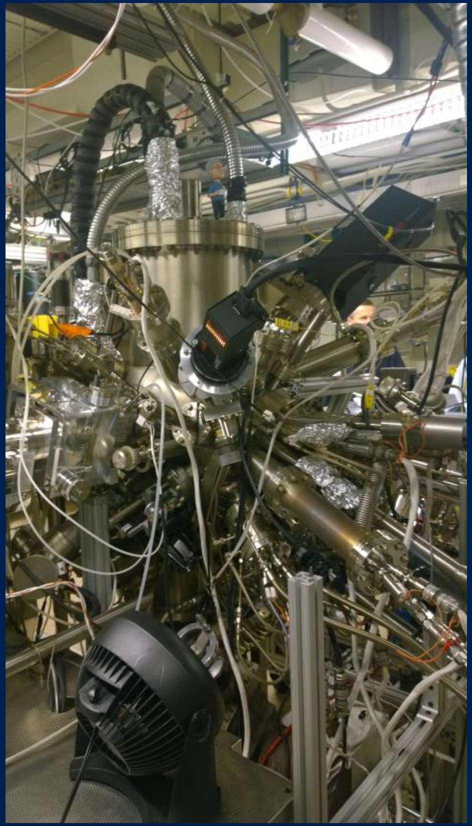
Martin Roetteler
Quantum Architectures and Computation Group (QuArC)
Microsoft Research

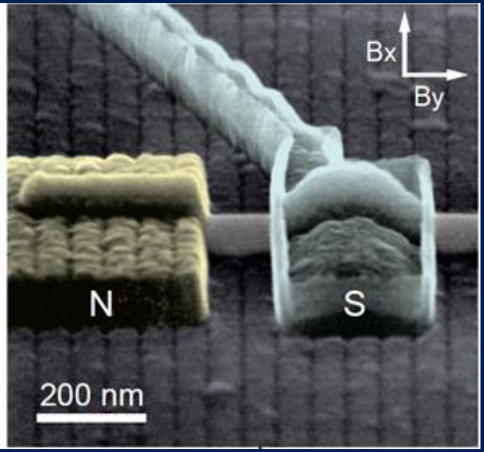
Northwest C++ Users' Group
Redmond, WA
June 15, 2016

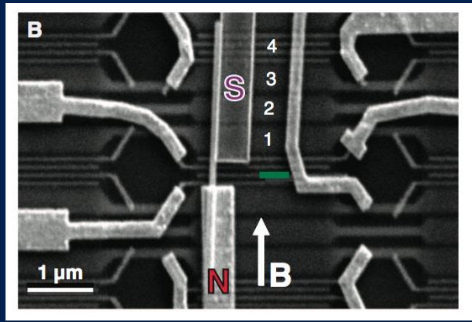
Microsoft QuArC and StationQ







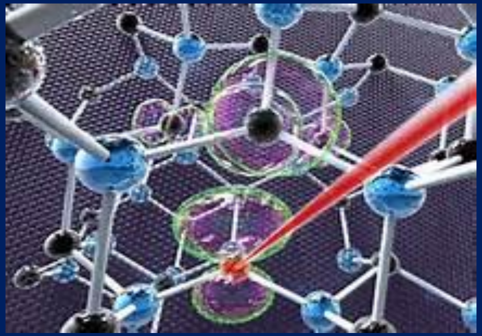




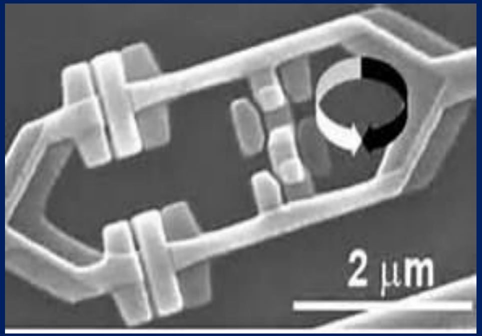
Quantum hardware technologies

Ion traps

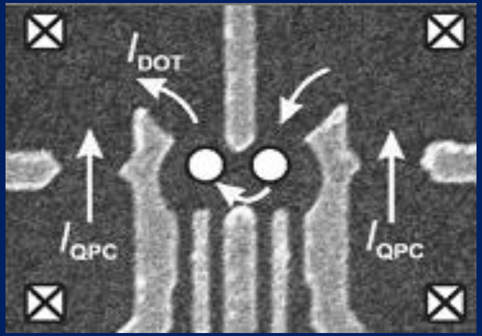
NV centers



Super-conductors

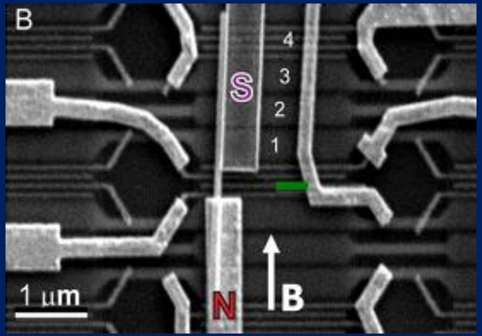


Quantum dots



Linear optics

Majorana zero modes



Charlie Marcus' Lab in Copenhagen

“Quantum killer apps”

Quantum algorithms

Shor's Algorithm (1994)

- Breaks RSA, elliptic curve signatures, DSA, El-Gamal
- Exponential speedups

Solving Linear Systems of Equations (2010)

- Applications shown for electromagnetic wave scattering
- Exponential speedups

Quantum Simulation (1982)

- Simulate physical systems in a quantum mechanical device
- Exponential speedups

Motivation: real-world use cases

Nitrogen Fixation

Efficiently convert nitrogen to fertilizer

100-200 qubits: Design catalysts to enable efficient fertilizer production

Carbon Capture

Capture carbon directly from the air at any location

100-200 qubits: Design catalysts to capture waste carbon with less energy

Materials Science

Find a material that superconducts at room temperature, organic batteries

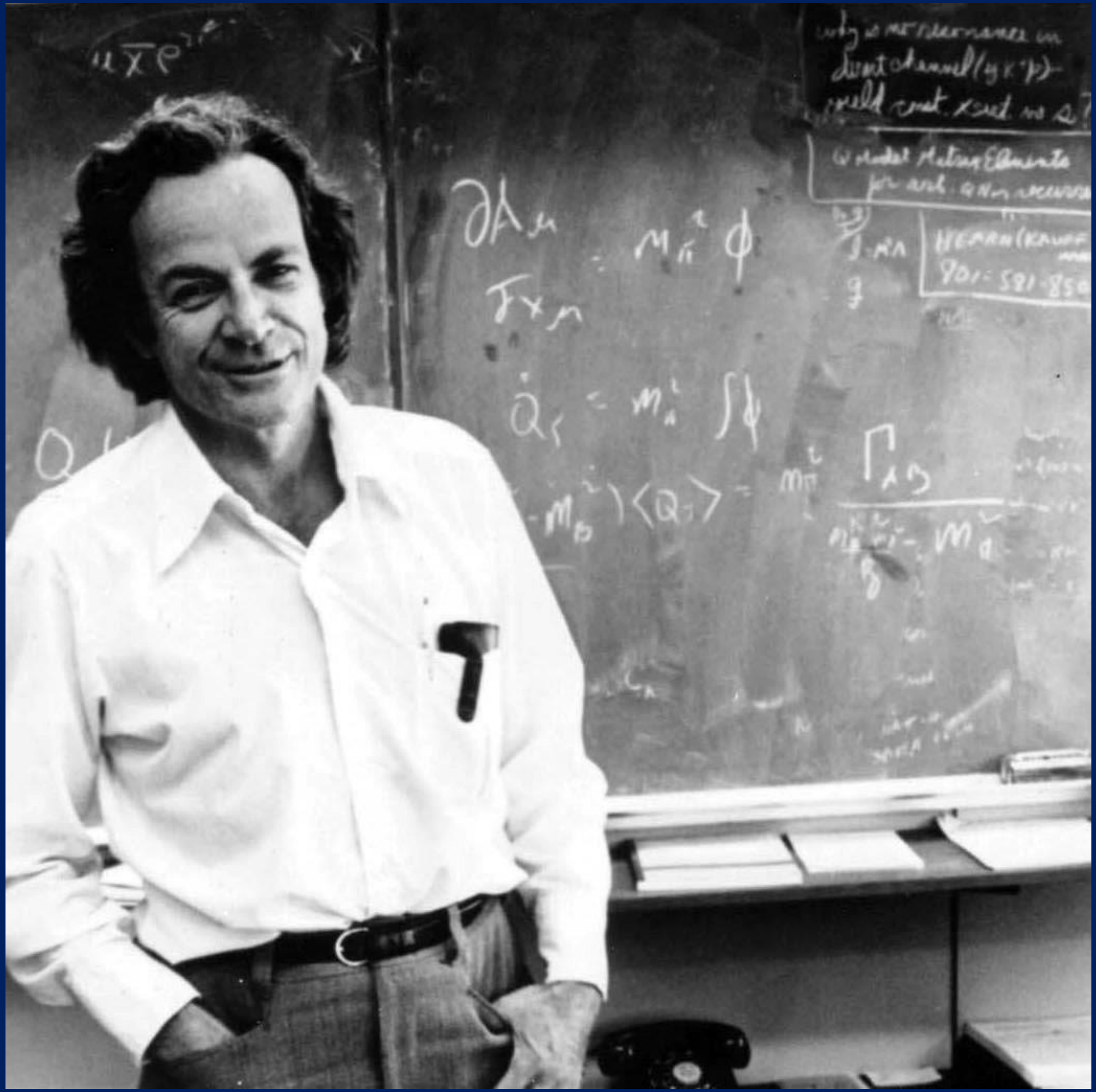
100s-1000s qubits: Simulate large systems in time linear in the number of particles

Machine Learning

Conventional learning uses approximations to train efficiently

100s-1000s qubits: Replace approximations with better solutions

Superposition

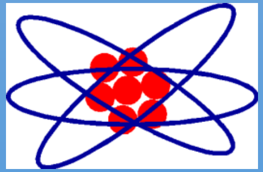


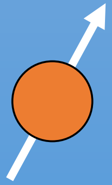
- Feynman taught us that the universe via quantum mechanics is the ultimate parallel computer
- Quantum mechanics considers all paths at once

Interference

Allows to cancel out useless computations and to amplify useful ones

Quantum Magic: Qubits and Superposition





single atom

$|g\rangle = |0\rangle$
$|e\rangle = |1\rangle$

single spin

$|\downarrow\rangle = |0\rangle$
$|\uparrow\rangle = |1\rangle$

$|\psi\rangle = |0\rangle$ … $\rangle = \downarrow + \uparrow$

Information encoded in the state of a two-level quantum system

Input

Output

P(1)

+ P(2)

⋮

+ P(2^n)

Q(1)

+ Q(2)

⋮

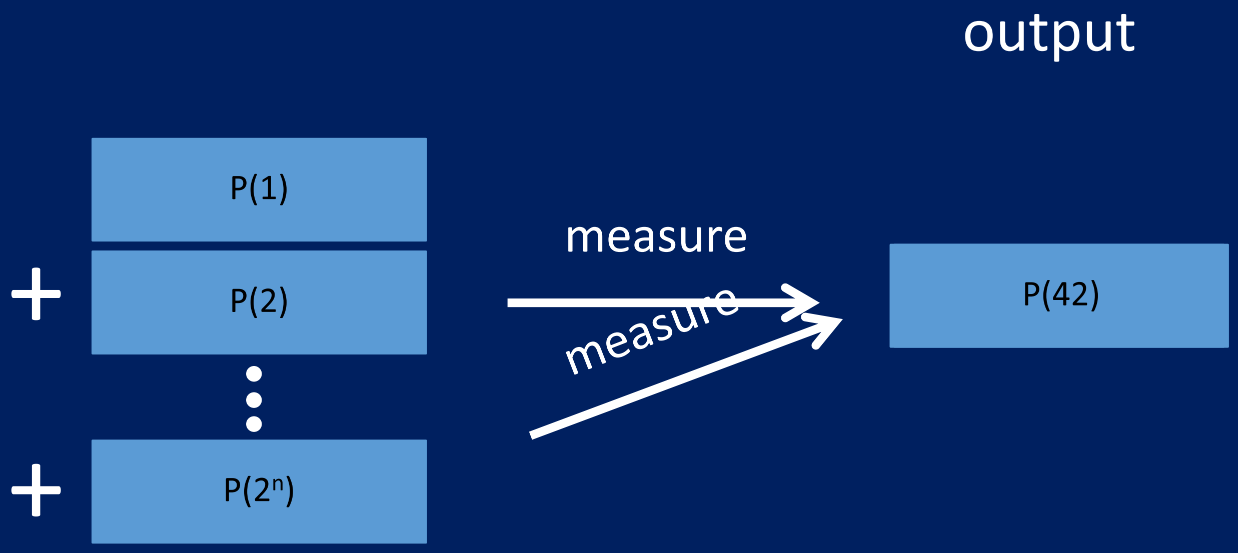
+ Q(2^n)

Any catches?

No-cloning principle

Quantum information cannot be copied

I/O limitations

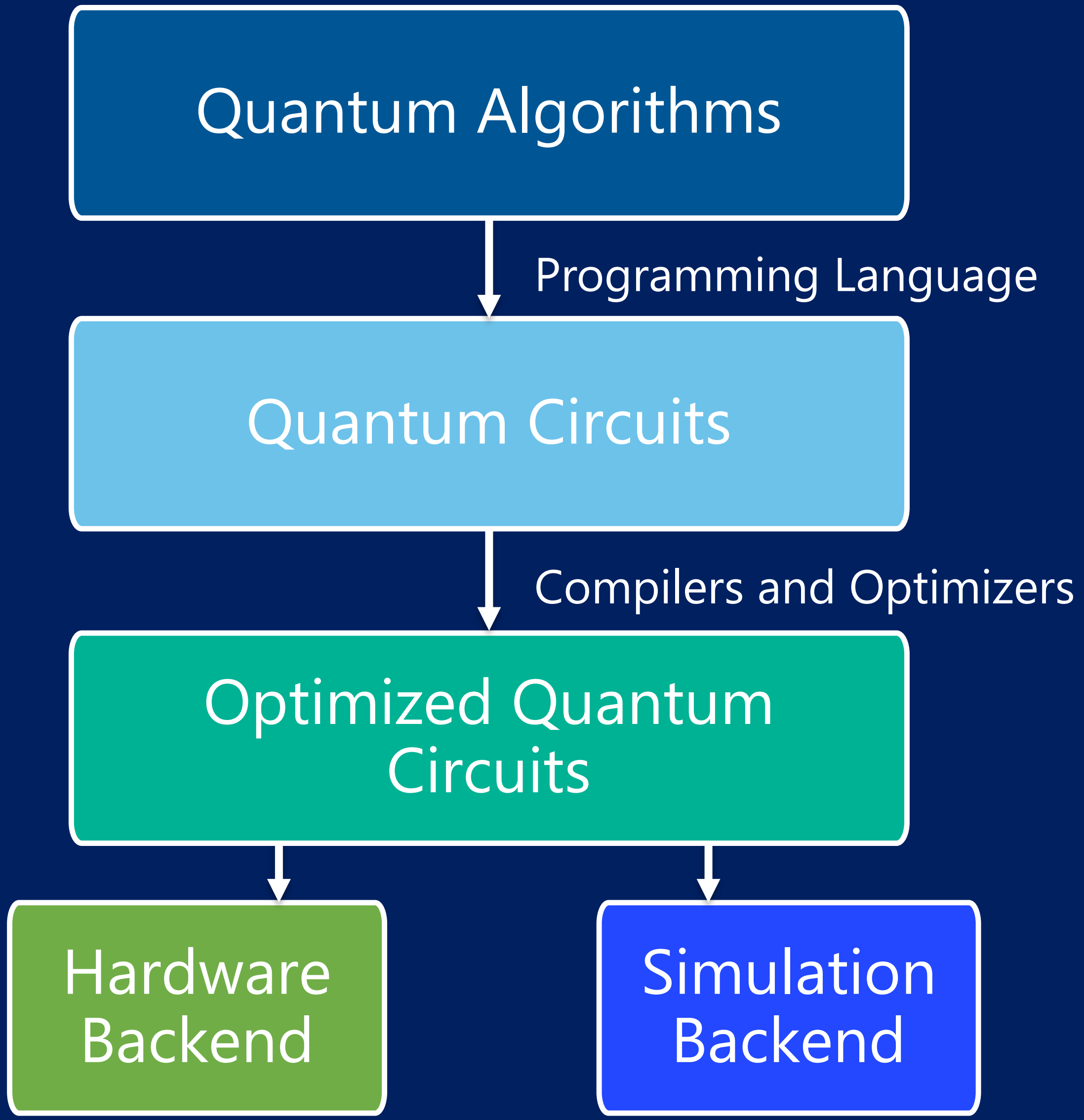


Input: preparing initial state can be costly

Output: reading out a state is probabilistic

How to program a quantum computer?

A Software Architecture for Quantum Computing



- Automatically maps a quantum algorithm to executable code for a quantum computer
- Increases speed of innovation
  - Rapid development of quantum algorithms
  - Efficient testing of architectural designs
  - Flexible for the future

The LIQ$Ui|\rangle$ platform

Wecker and Svore, 2014

LIQ$Ui|\rangle$ goals

- Simulation:
  - High enough level language to easily implement large quantum algorithms
  - Allow as large a simulation on classical computers as possible
  - Support abstraction and visualization to help the user
  - Implement as an extensible platform so users can tailor to their own requirements
- Compilation:
  - Multi-level analysis of circuits to allow many types of optimization
  - Circuit re-writing for specific needs (e.g., different gate sets, noise modeling)
  - Compilation into real target architectures

The $\text{LIQ}Ui|\rangle$ simulation platform

- We chos… …um algorithms
  - F# is als…
- Optimize…
  - Paralleli…
  - Many hi… …s growing a complex… …n
  - A CHP-… …ns that don't require full circu…
- Public re…
  - Restrict…
  - No software restrictions on the stabilizer simulator

LIQUi|>: A Software Design Architecture and Domain-Specific Language for Quantum Computing. Dave Wecker, Krysta M. Svore

Languages, compilers, and computer-aided design tools will be essential for scalable quantum computing, which promises an exponential leap in our ability to execute complex tasks. LIQUi|> is a modular software architecture designed to control quantum hardware. It enables easy programming, compilation, and simulation of quantum algorithms and circuits, and is independent of a specific quantum architecture. LIQUi|> contains an embedded, domain-specific language designed for programming quantum algorithms, with F# as the host language. It also allows the extraction of a circuit data structure that can be used for optimization, rendering, or translation. The circuit can also be exported to external hardware and software environments. Two different simulation environments are available to the user which allow a trade-off between number of qubits and class of operations. LIQUi|> has been implemented on a wide range of runtimes as back-ends with a single user front-end. We describe the significant components of the design architecture and how to express any given quantum algorithm.

Paper: http://arxiv.org/abs/1402.4467

Download: http://stationq.github.io/Liquid

LIQ$Ui|\rangle$ - Optimizations

- Basic definition: To operate on n qubits requires: $\mathrm{U}_{2^n,2^n} \times \Psi_{2^n}$
- We run out of simulation address space and physical memory very quickly
- State:
  - If we break the state up into pieces (sets of entangled qubits) then only the largest entangled “register” limits the computation
  - The state can’t be compressed further since it’s dense (in general) and must be represented with high precision.
- Operator:
  - Usually very sparse, but requires a large amount of bookkeeping and overhead to manipulate (inefficient) and is still as big or bigger than the state (even with massive compression)

LIQ$Ui|\rangle$ - Optimizations

- If we can guarantee that the qubits we want to operate on are always at the beginning of the state vector, we can view the operation as:

$$G_{2^k,2^k} \otimes I_{2^{n-k},2^{n-k}} \times \Psi_{2^n}$$

- However, what we'd really like is to flip the Kronecker product order:

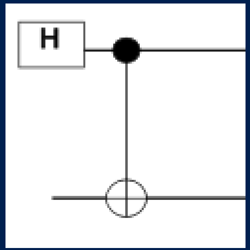
$$I_{2^{n-k},2^{n-k}} \otimes G_{2^k,2^k} \times \Psi_{2^n}$$

- This accomplishes :
  - $I \otimes G$ becomes a block diagonal matrix that just has copies of G down the diagonal. This means that you'd never have to actually materialize $U = I \otimes G$
  - Processing is highly parallel (and/or distributed) because the matrix is perfectly partitioned and applies to separate, independent parts of the state vector

Quantum "Hello World!"

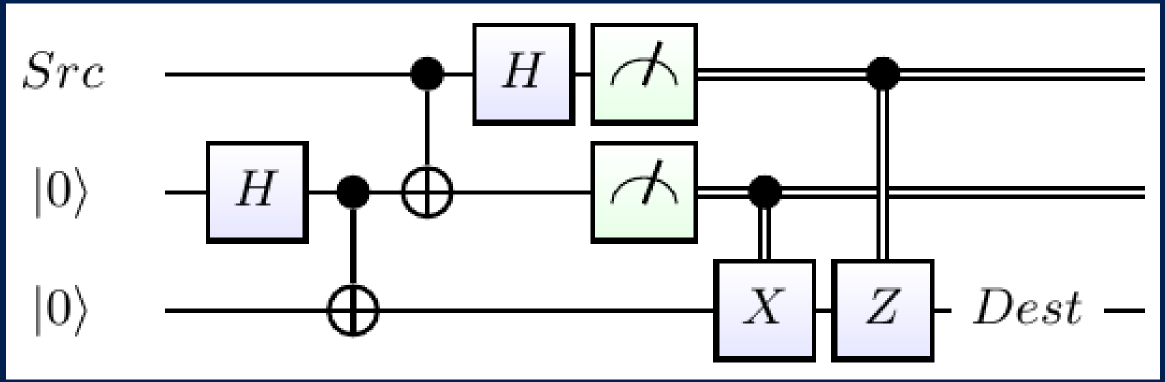
- Define a function to generate entanglement:

```
let EPR (qs:Qubits) = H qs; CNOT qs
```

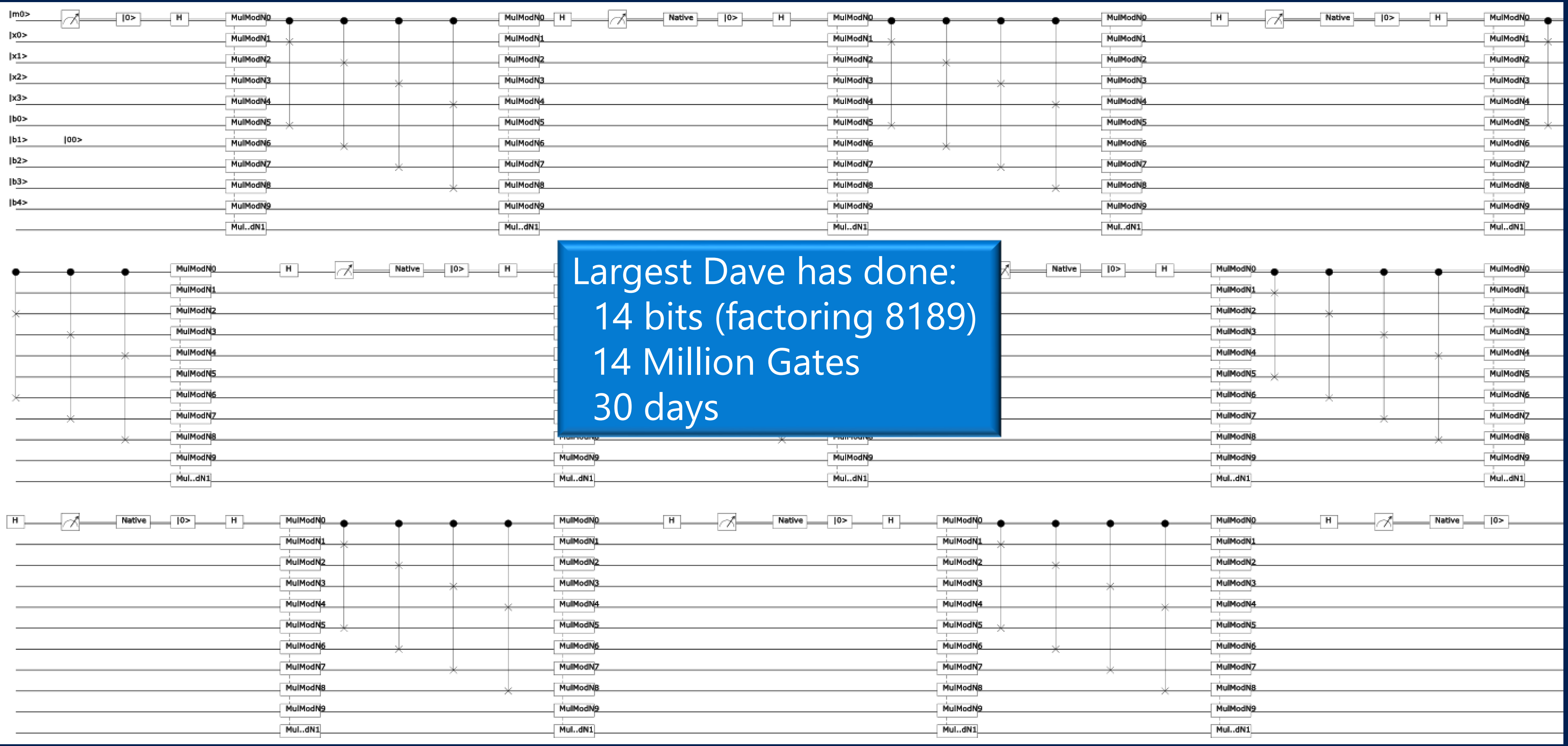


- The rest of the algorithm:

```
let teleport (qs:Qubits) =
    let qs'      = qs.Tail
    EPR qs'; CNOT qs; H qs
    M qs'; BC X qs'
    M qs ; BC Z !!(qs,0,2)
```



Shor's algorithm: full circuit: 4 bits ≅ 8200 gates



Circuit for Shor's algorithm using 2n+3 qubits – Stéphane Beauregard

Obtain the package from:
http://stationq.github.io/Liquid

Microsoft's quantum computing group:
http://research.microsoft.com/groups/quarc/
http://research.microsoft.com/en-us/labs/stationq/

Microsoft

martinro@microsoft.com