Solace of Quantum

July 14, 2022

Henry Dietz
Electrical & Computer Engineering

Aggregate.Org UNBRIDLED COMPUTING

University of Kentucky

Solace of Quantum

We live in what appears to be the end days of Moore's Law. His 1965 prediction that the amount of circuitry one could cost-effectively place on a chip would exponentially double roughly every two years meant that computers could become faster primarily by parallel processing: performing many operations simultaneously. Unfortunately, now that rate seems to have slowed and we also are suffering from the fact that power consumption per unit circuitry has not dropped as fast as circuit complexity has grown. What we need is a way to continue to increase parallelism without correspondingly increasing the amount of circuitry needed and power consumed.

Quantum computing has the potential to provide a very restrictive, but massively parallel, form of computation in which an exponential amount of parallel computation is supported per unit hardware and power consumption. This talk will focus mostly on how quantum computing works as a computational model: how are they are programmed and what they can -- and cannot -- do. The current state of the art in quantum computing, and what the future might bring, will be discussed.

Aggregate.Org UNBRIDLED COMPUTING
University of Kentucky


QUANTUM OF
SOLACE

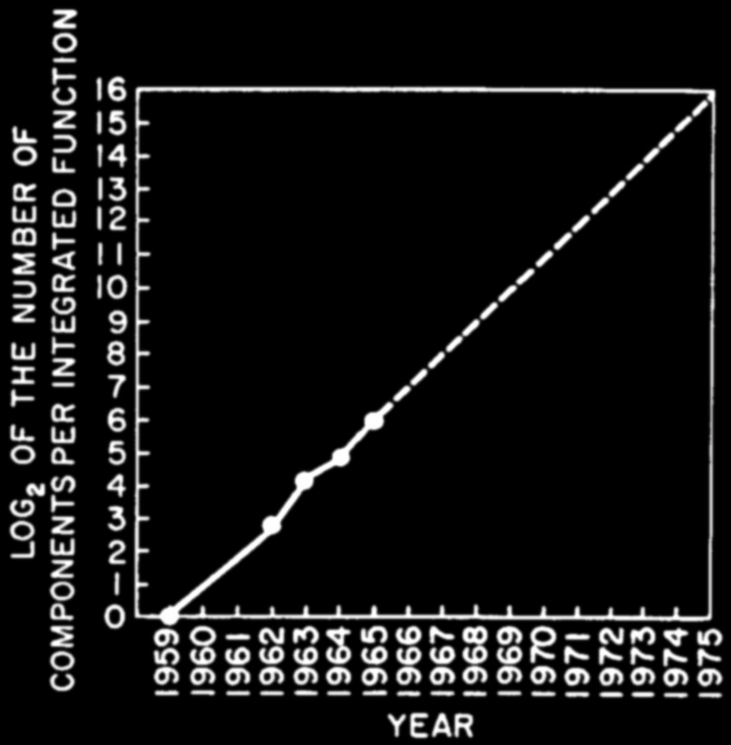
Solace of Quantum?

Solace: *comfort or consolation in a time of distress or sadness*

- What are we so upset about?
- How is **Quantum Computing** comforting us?

How Computers Get Faster: Moore's Law

- 1965 prediction
  - Not about chip speed
  - Circuit complexity 2X every 18-24 months
- Speedup is mostly about **parallel processing**

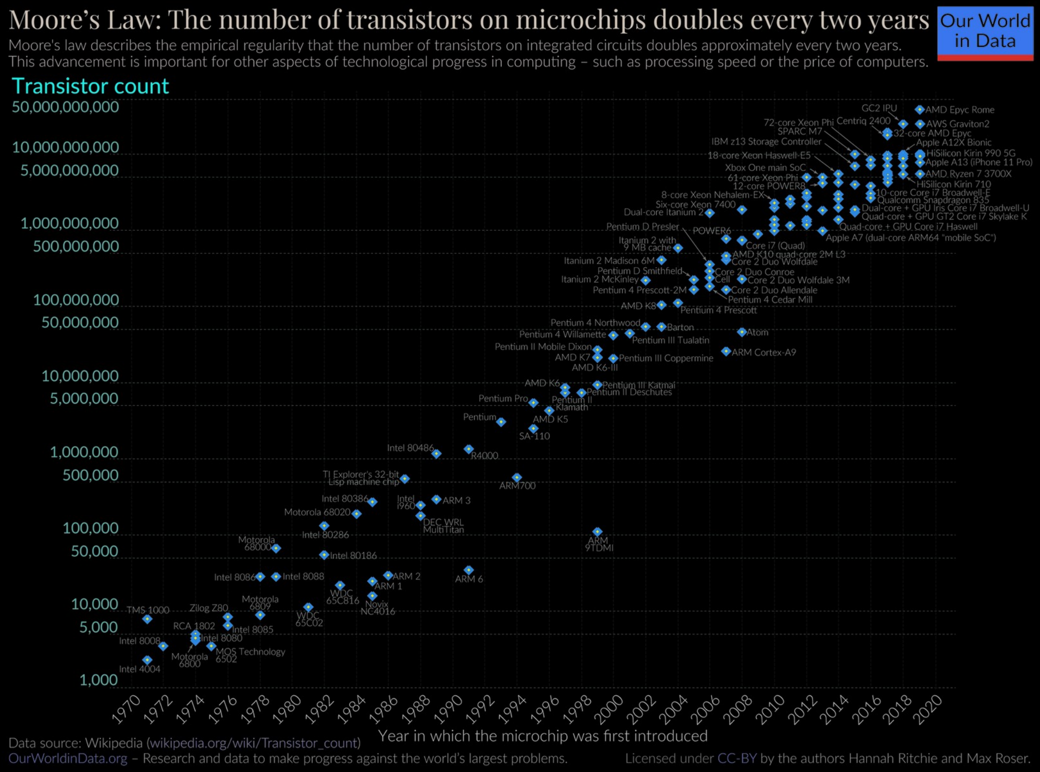


Parallel Processing

- Break program into ***N*** pieces that can execute simultaneously
  - **Scalable**: bigger ***N***, more speedup
  - **Modular hardware**
  - Can be **fault tolerant** using redundancy
- Massive parallelism makes big, power-hungry, computers if Moore's Law fails us...

Is Moore’s Law Still Valid?

- **It was**…
  - Until ~2013
  - Not dead yet
  - A little slow now, not recovering?
- **It *will* end.**

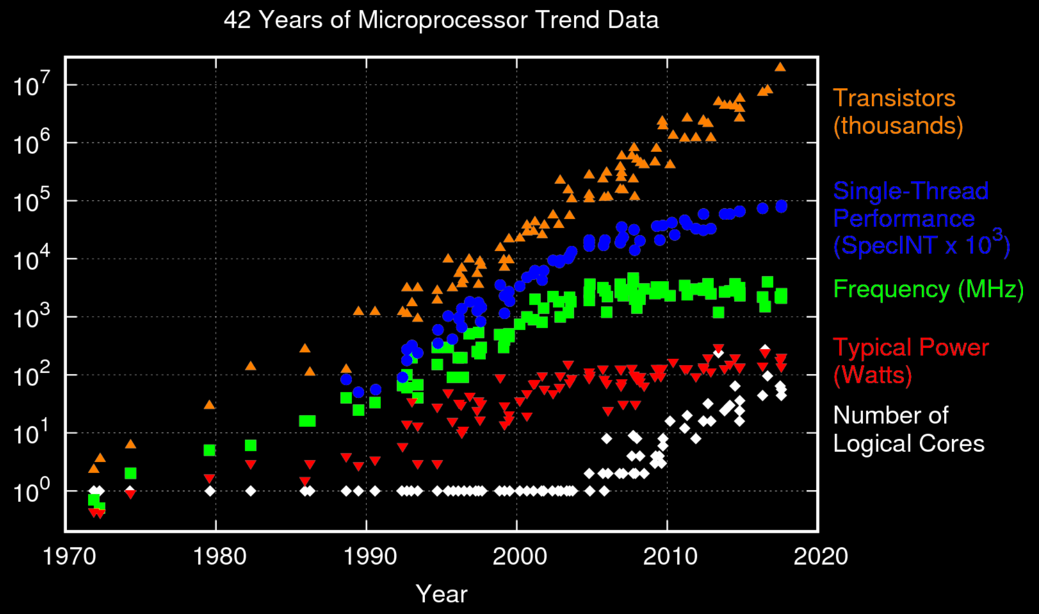


Why We Are Upset: Top500.Org

- Tracking supercomputer speed since 1993
  - 2X every year (faster than ML!)
  - Since 2013, curves are leveling-off...
- What's happening?

All The Bad News

- ML slowing
- Power/transistor ▼ slower than transistors/chip ▲
- Individual ops not getting much faster



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Solace of Quantum Computing

- Massively-parallel processing ***without*** massively parallel hardware
- Potentially very low power consumption per unit computation performed
- Individual ops could be almost instantaneous

Quantum Computing

- State-of-the-art conventional processor chips already depend on quantum phenomena, but just implement conventional logic with it
- **Quantum Computing** is about using quantum phenomena to ***implement a different model***
  - Adiabatic optimization (e.g., by DWave)
  - **Quantum gates** (by most others)

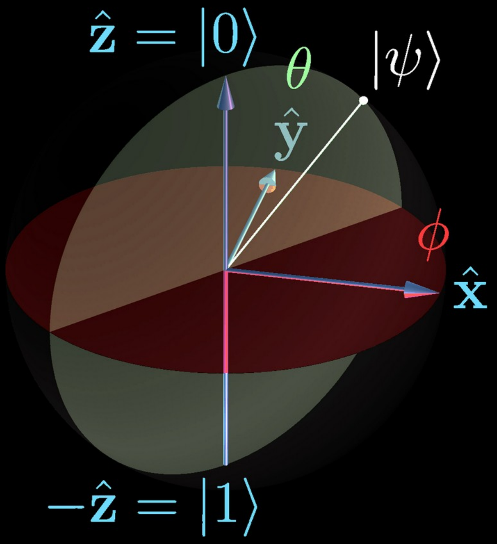
Conventional Computing

- **Memory** is made of **Bits**, each holding 0 or 1
  - Bit values reliably persist ***forever***
  - **Every bit** can be accessed by addressing
- **Processor** (perhaps one of many in a system)
  - Gates: **`AND`**, **`OR`**, **`XOR`**, **`NOT`**, **`NAND`**, **`NOR`**, **`MUX`**…
  - **Fanout** is allowed (e.g., FOF = fanout of 4)

Quantum Computing

- **Memory** is made of **Qubits**, each holding a ***probability density function*** for 0 and 1
  - Qubit value collapses to 0 or 1 when read
  - Values have a limited lifespan (decoherence)
- **Processor** (really PIM: processors in memory)
  - Gates: **`NOT`**, **`CNOT`**, **`CCNOT`**, **`SWAP`**, **`CSWAP`**…
  - **Fanout** is **not** allowed

Bloch Sphere Qubit Model



$$|\psi\rangle = \cos(\theta/2)|0\rangle + e^{i\phi}\sin(\theta/2)|1\rangle$$

$$= \cos(\theta/2)|0\rangle + (\cos\phi + i\sin\phi)\sin(\theta/2)|1\rangle$$

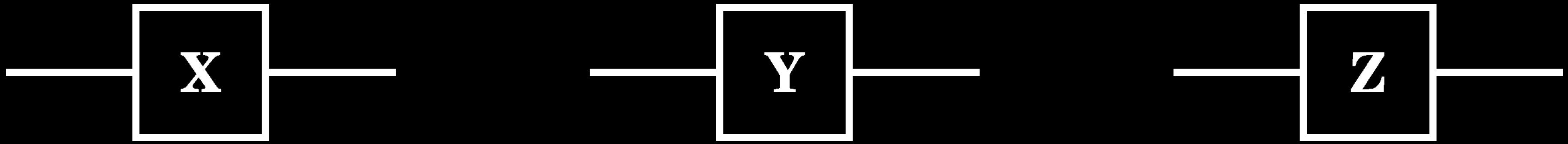
where $0 \leq \theta \leq \pi$ and $0 \leq \phi < 2\pi$

- Value of a Qubit is a ***wave function***
- Probability by coordinates on sphere surface

Quantum Processor

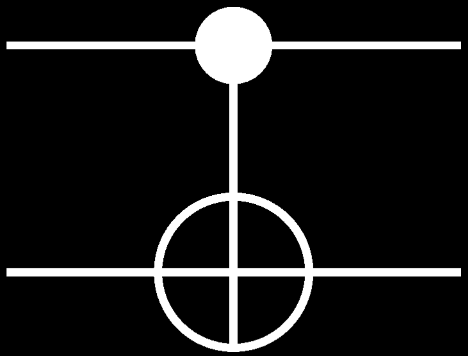
- Gates aren’t hardware structures
  - Gates operate on Qubits “in place”
  - Gates are forces imposed on Qubits
  - Conventional computer implements control
- **Processor** (in the best case, one per qubit)
  - Gates: **`NOT`**, **`CNOT`**, **`CCNOT`**, **`SWAP`**, **`CSWAP`**… all must be **thermodynamically reversible**
  - **Fanout** is **not** allowed

Quantum Gate Types: Pauli



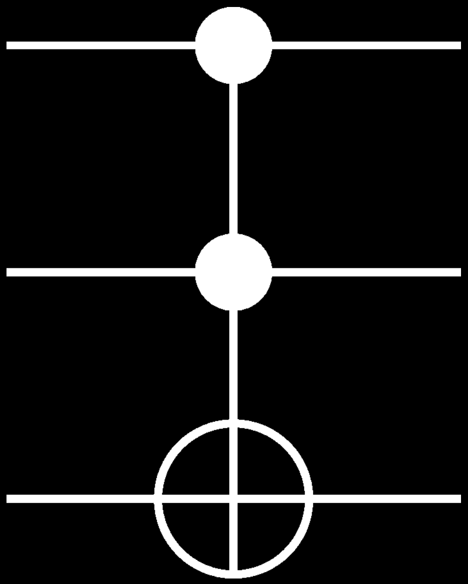
- **Pauli `X`** is also known as **`NOT`**
  - Rotates Bloch Sphere around X by π radians
  - Functions like conventional **`NOT`**
  - **`NOT`** is its own inverse
- **Pauli `Y`** rotates around Y and **Pauli `Z`** around Z

Quantum Gate Types: CNOT



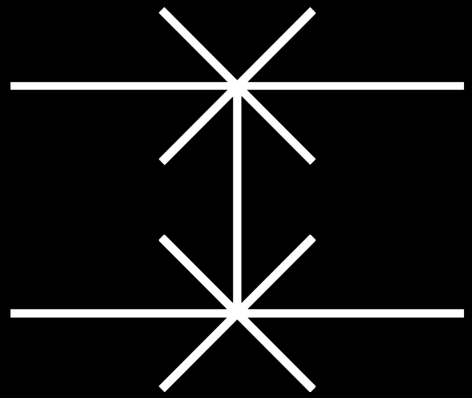
- CNOT is the **Controlled NOT** gate
  - Top input is control, passes thru unchanged
  - Bottom input is inverted where control is 1
  - Both inputs can’t be the same Qubit
  - Similar to conventional XOR gate

Quantum Gate Types: Toffoli



- **Toffoli** is also known as **`CCNOT`**, **Controlled Controlled `NOT`**
  - A classical universal gate
  - Top two inputs pass unchanged
  - Bottom input is inverted where both control inputs are 1
  - Behaves like `C = (A AND B) XOR C`

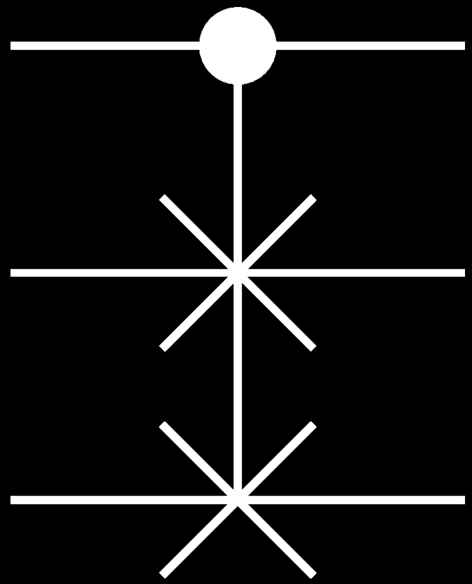
Quantum Gate Types: SWAP



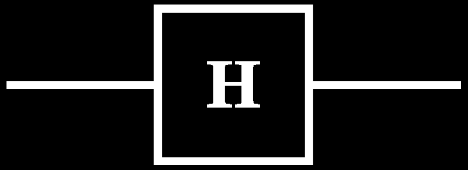
- **SWAP** exchanges values of two Qubits
  - Seems pointless...
    but this is a reversible assignment

Quantum Gate Types: Fredkin

- **Fredkin** is also known as **`CSWAP`**, **Controlled `SWAP`**
  - A classical universal gate… and *billiard-ball* conservative
  - Top input passes unchanged
  - Bottom inputs are swapped where top control input is 1
  - Behaves like paired conventional `MUX`es



Quantum Gate Types: Hadamard



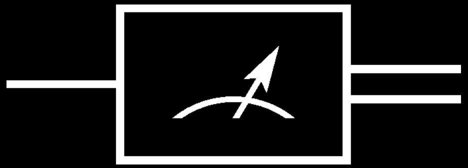
- **Hadamard** is not like any conventional gate
  - A Qubit can only be initialized to 0 or 1
  - Hadamard operator converts that into the equiprobable superposed state: 50% 0, 50% 1
- If *applied in parallel* to ***E*** Qubits, the result is the equiprobable ***E***-way entangled superposition

Equiprobable *E*-Way Entangled Superposition?

- Up to this point, nothing about Quantum Computing sounded better than conventional…
- Suppose we apply **H** in parallel to 16 Qubits?
  - Those 16 Qubits will hold all 65,536 possible 16-bit values with equal probabilities
  - Any single operation on any of those Qubits will effectively operate on all 65,536 values

Parallel processing without parallel hardware!

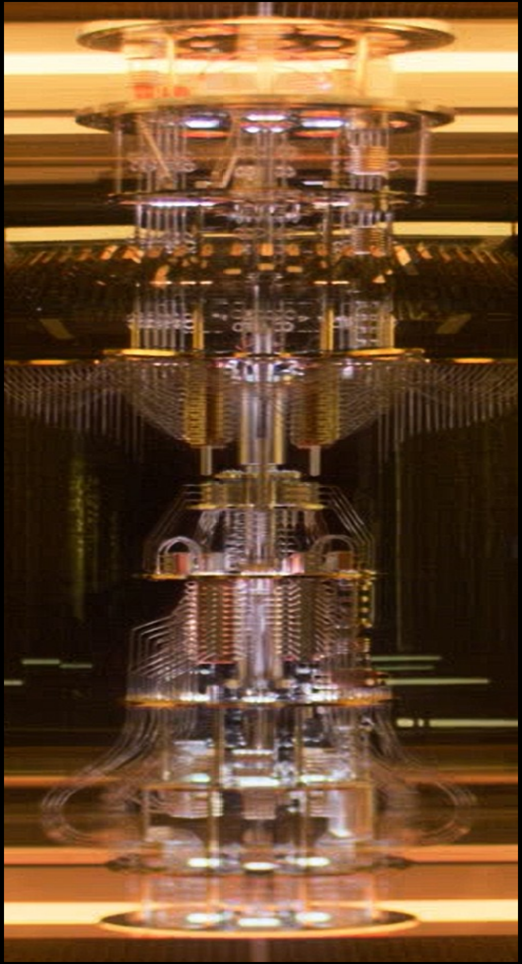
Quantum Gate Types: **Measurement**



- **Measurement** collapses a superposition
  - Superposed Qubit becomes either 0 or 1
  - Superposed probability density function is randomly sampled, determines odds of 0 vs. 1

Exponentially cheap parallel computation…
but you only get to read-out one answer per run

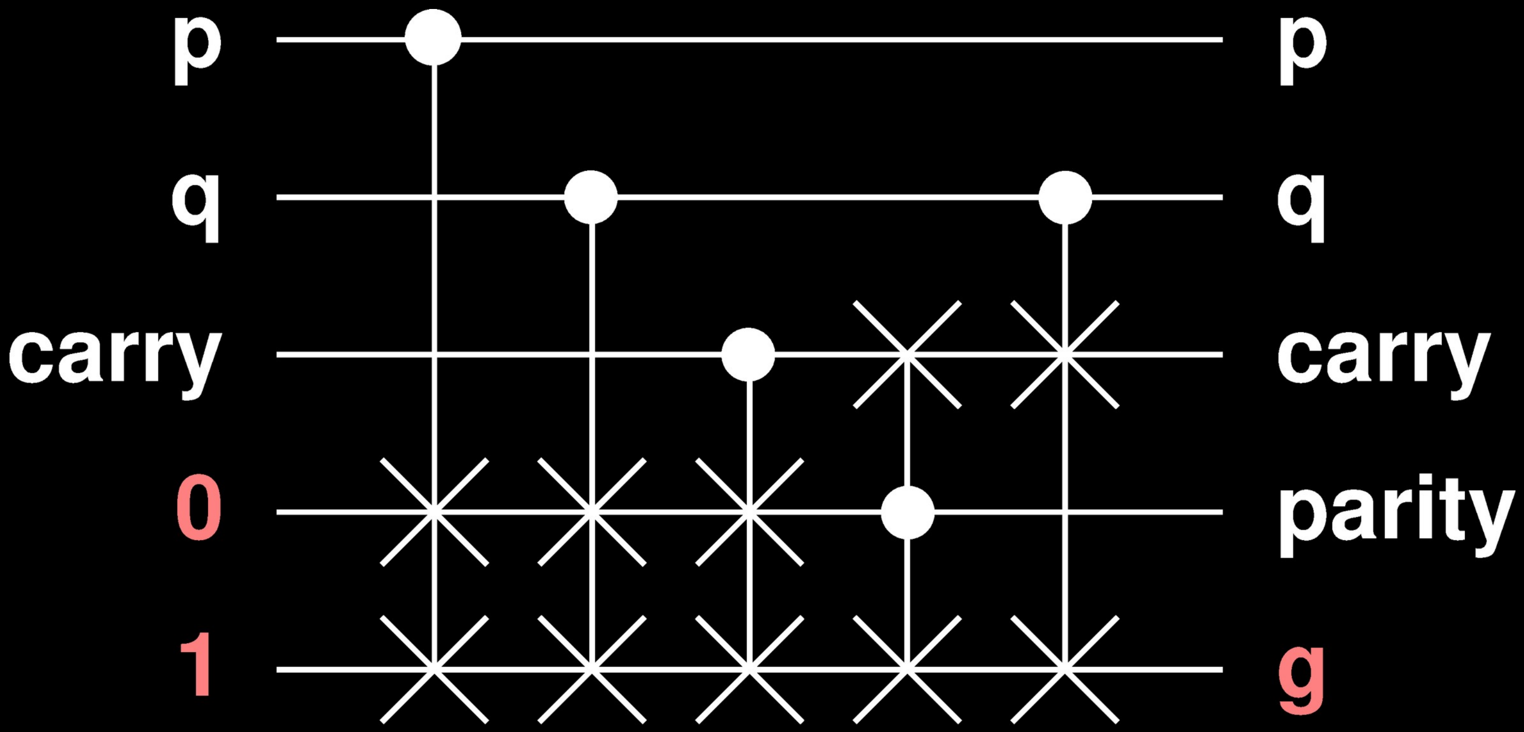
Quantum Computers?



A Real Example On A Fake Computer

- **KREQC**: Kentucky's Rotationally Emulated Quantum Computer
  - 2018: 6 “Q-bits” 6-way entangled
  - 2019: 16 “Q-bits” 16-way entangled
- Really just a display for a simulator; each “Q-bit” shows probability of 0 vs. 1 by angle of core

Let’s Build A 1-Bit Full Adder



{carry, parity} = p + q + carry

Let’s Build A 1-Bit Full Adder

KREQC Program

```
// 1-bit full adder
p=1;
q=1;
carry=0;
parity=0;
g=1;
CSWAP(p, parity, g);
CSWAP(q, parity, g);
CSWAP(carry, parity, g);
CSWAP(parity, carry, g);
CSWAP(q, carry, g);
```

Simulation Output

```
QUBIT                  g  parity   carry       q       p
            32 |    64 |     0 |     0 |    64 |    64 |
CSWAP          |       x-------x-------|-------|-------@
            32 |     0 |    64 |     0 |    64 |    64 |
CSWAP          |       x-------x-------|-------@       |
            32 |    64 |     0 |     0 |    64 |    64 |
CSWAP          |       x-------x-------@       |       |
            32 |    64 |     0 |     0 |    64 |    64 |
CSWAP          |       x-------@-------x       |       |
            32 |    64 |     0 |     0 |    64 |    64 |
CSWAP          |       x-------|-------x-------@       |
            32 |     0 |     0 |    64 |    64 |    64 |
               1       0       0       1       1       1


                       g  parity   carry       q       p
   64/64               0       0       1       1       1
```

Now Give It Superposed Input

KREQC Program

```
// 1-bit full adder
p=1;
q=0;
carry=?;
parity=0;
g=1;
CSWAP(p, parity, g);
CSWAP(q, parity, g);
CSWAP(carry, parity, g);
CSWAP(parity, carry, g);
CSWAP(q, carry, g);
```

Simulation Output

```
QUBIT                      g   parity     carry         q        p
              32 |      64 |       0 |      32 |       0 |      64 |
CSWAP            |         x-------x-------|-------|-------@
              32 |       0 |      64 |      32 |       0 |      64 |
CSWAP            |         x-------x-------|-------@        |
              32 |       0 |      64 |      32 |       0 |      64 |
CSWAP            |         x-------x-------@        |        |
              32 |      32 |      32 |      32 |       0 |      64 |
CSWAP            |         x-------@-------x        |        |
              32 |      32 |      32 |      32 |       0 |      64 |
CSWAP            |         x-------|-------x-------@        |
              32 |      32 |      32 |      32 |       0 |      64 |
                 0         1       0       1       0       1

                           g   parity     carry         q        p
     32/64                 0        1         0         0        1
     32/64                 1        0         1         0        1
```

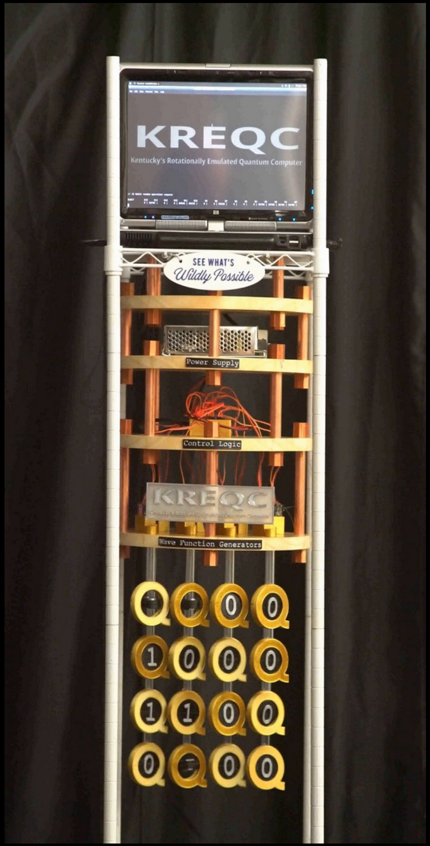
KREQC Program

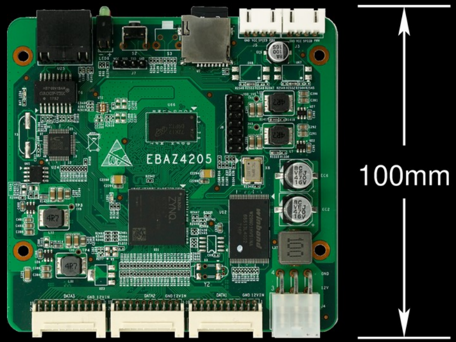
```
// 1-bit full adder
p=?;
q=?;
carry=?;
parity=0;
g=1;
CSWAP(p, parity, g);
CSWAP(q, parity, g);
CSWAP(carry, parity, g);
CSWAP(parity, carry, g);
CSWAP(q, carry, g);
```

Simulation Output

```
QUBIT                        g  parity   carry       q       p
              32 |      64 |       0 |      32 |      32 |      32 |
CSWAP            |         x-------x-------|-------|-------@
              32 |      32 |      32 |      32 |      32 |      32 |
CSWAP            |         x-------x-------|-------@       |
              32 |      32 |      32 |      32 |      32 |      32 |
CSWAP            |         x-------x-------@       |       |
              32 |      32 |      32 |      32 |      32 |      32 |
CSWAP            |         x-------@-------x       |       |
              32 |      48 |      32 |      16 |      32 |      32 |
CSWAP            |         x-------|-------x-------@       |
              32 |      32 |      32 |      32 |      32 |      32 |
                 1         1       0       0       0       0
```

	g	parity	carry	q	p
8/64	0	0	1	1	1
8/64	0	1	0	0	1
8/64	0	1	0	1	0
8/64	0	1	1	1	1
8/64	1	0	0	0	0
8/64	1	0	1	0	1
8/64	1	0	1	1	0
8/64	1	1	0	0	0


KREQC
Kentucky's Rotationally Emulated Quantum Computer
SEE WHAT'S
Wildly Possible
Power Supply
Control Logic
KREQC
Wave Function Generators


EBAZ4205
100mm

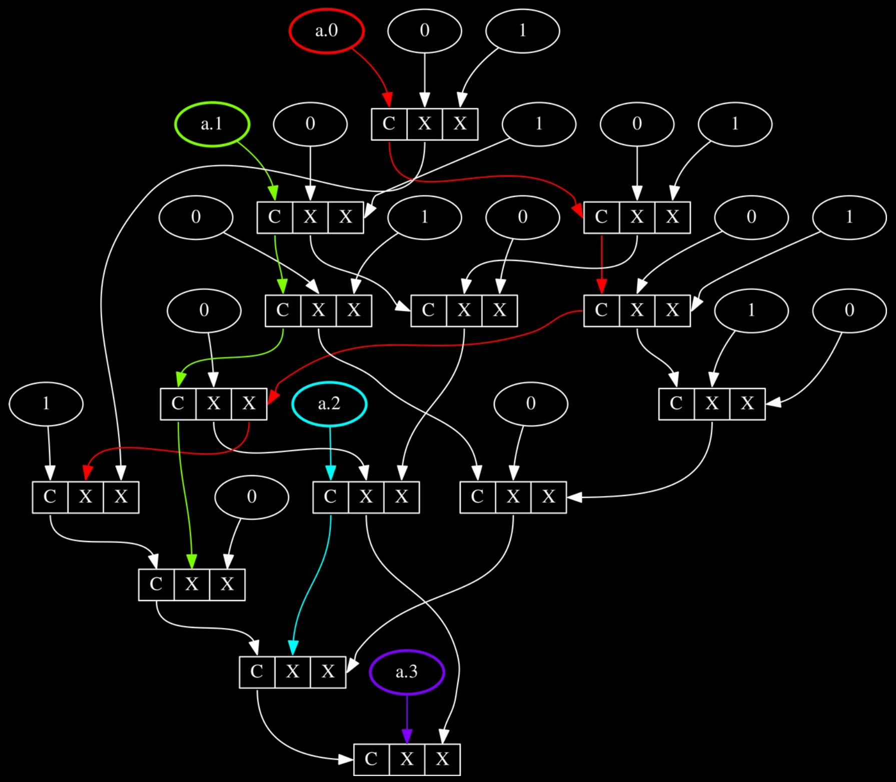
So, What Is Quantum Good For?

- Problems where:
  - You need to try all possible values
  - You don't need all answers, just one or some†
  - You don't mind occasionally wrong answers
  - Combinatorial logic operating on few qubits

† KREQC uses **pbits** (pattern bits), rather than qubits, for entangled superposition... that's why it can list all values and precise probabilities

int:4 a; a=a*a;

```
0:    -----X---X---X
0:    ------X-------
0:    --X---C-------
0:    -------X----X-
0:    ----X--C------
0:    -X------XX----
0:    --------X-----
0:    ---X----C-----
0:    X---------X---
0:    -----------X--
1:    ------XX------
1:    --X-----------
1:    ----X---------
1:    -X------------
1:    ---X----------
1:    ----------CCCC
1:    X-------------
a.0:  CCC--X----X---
a.1:  ---CCC-----X--
a.2:  ---------C--X-
a.3:  -------------X
```



Quantum Supremacy or Advantage

Solving a useful problem faster than any classical computer could

- 2019 Google’s 53-qubit **Sycamore**
- 2020 China’s 113-qubit **Jiuzhang**
- 2021 IBM’s 127-qubit **Eagle**

Quantum Performance Metrics

- **Scale**: number of Qubits
- **Quantum Volume**: largest square-shaped circuit successfully implementable (>97.5% correct)
- **CLOPS**: circuit-layer operations per second

The biggest problem is **decoherence**, collapse of superposition due to noise. Higher entanglement gives exponential performance improvement, but also much greater noise sensitivity.

Solace?

- **Yeah!**
  - Higher entanglement $\Rightarrow$ exponential savings
  - Saves on storage & active gates/computation
- However, thus far:
  - More Qubits than you can entangle
  - Superpositions don’t survive many gate ops
  - Few Qubits, slow cycle times
- **Quantum computers are special-purpose**

Is Special-Purpose Bad?

- Attached special-purpose accelerators are common; for example: modern **GPUs**
- Having special-purpose “dark silicon” power-up when needed has allowed processor performance to keep improving within a fixed power budget
- Even **Shor’s Algorithm** to factor large numbers is *mostly conventional code*, but it repeatedly invokes a quantum period-finding subroutine...

Conclusions

- Quantum computing *is* a way past Moore's Law, for very specific types of computations
- Quantum computers still have a long way to go
  - Quantum hardware *might* never get there
  - Thinking about quantum algorithms often yields faster algorithms for conventional computers
- My Parallel Bit Pattern (PBP) uses conventional gates to provide quantum-like properties

A Bit About My Work...

- I built the world's 1st Linux cluster supercomputer, the model nearly all supercomputers now follow... but power/computation is too high at scale!
- My machine room (108A Marksbury) has 170kW, 30 tons air conditioning, heats half the building, and couldn't power one of thousands of racks in the current top supercomputers!

To Minimize Power/Computation

- Aggressively optimize code at the gate level
- Work only on active bits
- Try to leverage entangled superposition

I’ve created a model, **Parallel Bit Pattern** computing (**PBP**), that implements entangled superposition with conventional hardware and symbolic computation on compressed data