QDB: From Quantum Algorithms Towards Correct Quantum Programs

PRINCETON UNIVERSITY

Yipeng Huang, Margaret Martonosi

EPiQC
An NSF Expedition in Computing

Detailed debugging effort across quantum algorithms

Quantum chemistry algorithms

- Calculating molecule properties from first principles
- Use quantum mechanical system to simulate quantum mechanics!
- Near term: needs few qubits, needs no error correction

Shor's integer factorization quantum algorithm

- Factors large integers in polynomial time!
- (known best classical algorithms take exponential time)
- Distant future: needs many qubits, needs error correction

Where possible, validate across quantum languages

Semantic gap

- Need languages, abstractions…

Tools gap

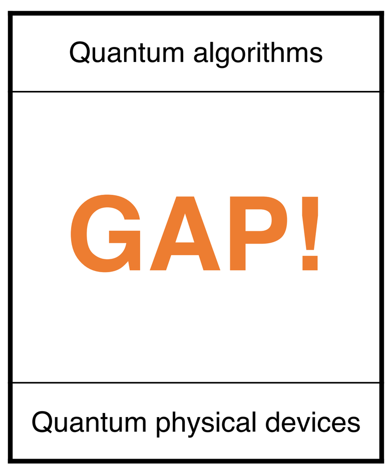
- Need optimizing compilers, simulators, debuggers…

Infrastructure gap

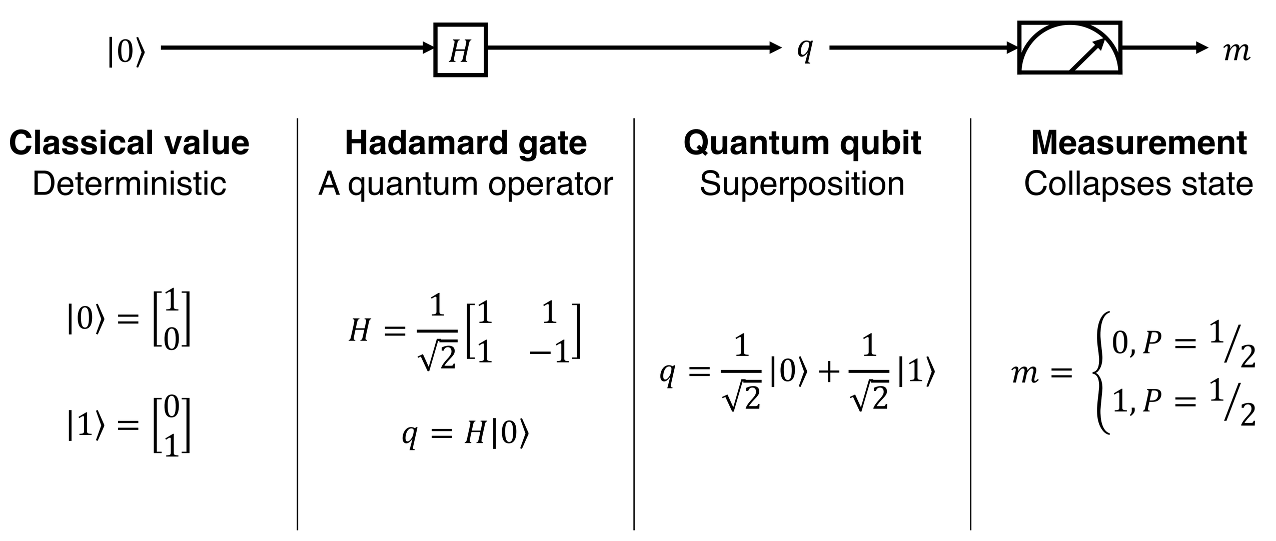
- Need more abundant, more reliable qubits…

Educational gap

- Need researchers, college curricula, K-12 pipeline…



Superposition underlies power, but precludes 'printf'



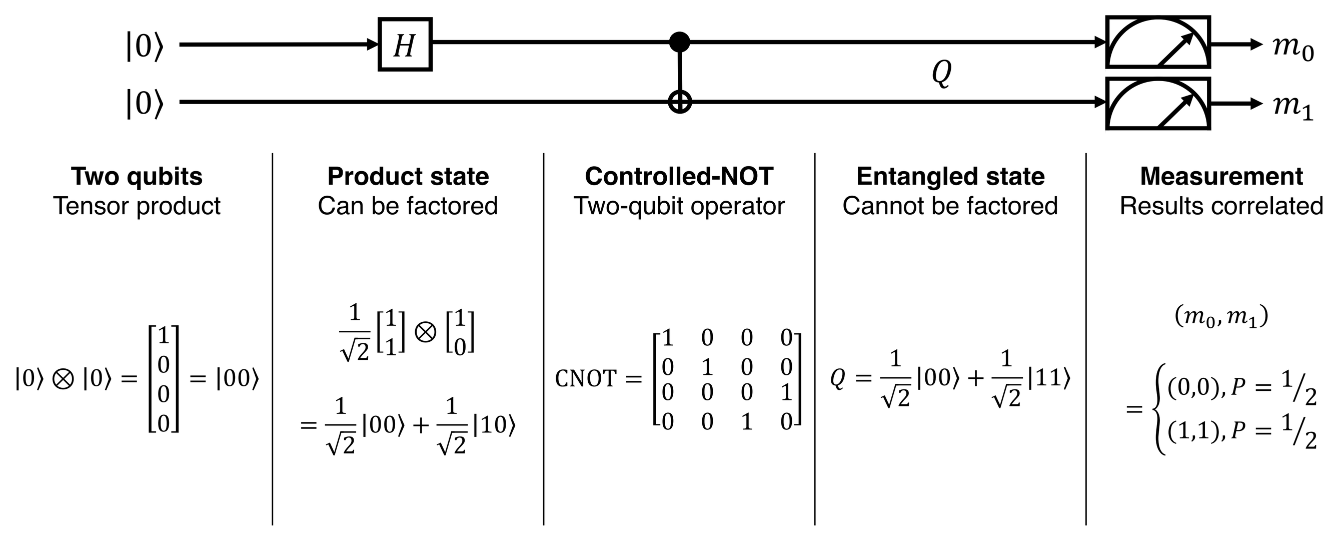
$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$, $|1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$

$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$, $q = H|0\rangle$

$q = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$

$m = \begin{cases}0, P = 1/2\\1, P = 1/2\end{cases}$

Huge state space limits simulation to 'toy' problems



$|0\rangle \otimes |0\rangle = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} = |00\rangle$

$\frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} \otimes \begin{bmatrix}1\\0\end{bmatrix} = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle$

$CNOT = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}$

$Q = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$

$(m_0, m_1) = \begin{cases}(0,0), P = 1/2\\(1,1), P = 1/2\end{cases}$

Teams now racing towards accurate and more qubits

Superconducting qubits: IBM, Google, Intel, Rigetti

Trapped ion qubits: University of Maryland / IonQ

Classify quantum programming bugs, pair with defenses, debugging and assertions

Bug type 1: classical input parameters

k, the algorithm iteration	0	1	2	3	…
$a = 7^{2^k} \bmod 15$	7	4	1	1	…
$a^{-1}; a \times a^{-1} \equiv 1 \bmod 15$	~~13~~ 12	4	1	1	…

Defense 1: algorithm progress assertions

Bug type 2: quantum initial values

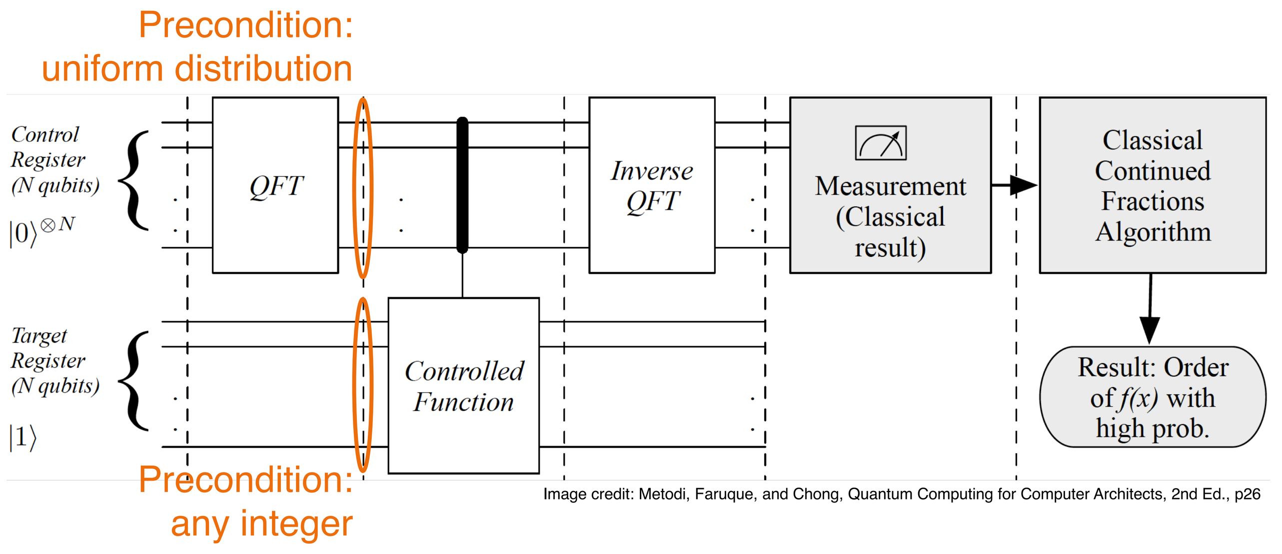
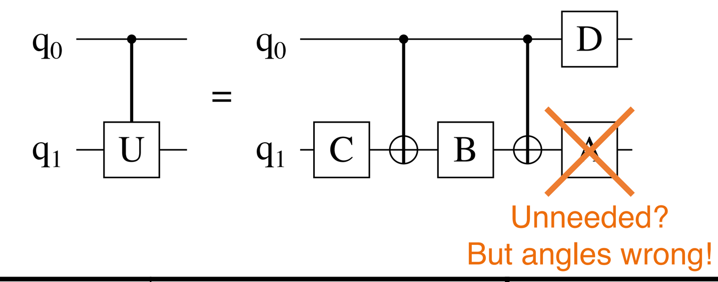


Image credit: Metodi, Faruque, and Chong, Quantum Computing for Computer Architects, 2nd Ed., p26

Defense 2: pre-condition assertions

Bug type 3: coding up basic operations




```
Rz(q1, +angle/2); // C
CNOT(q0, q1);
Rz(q1, -angle/2); // B
CNOT(q0, q1);
Rz(q0, +angle/2); // D
```

Correct, operation A unneeded

```
CNOT(q0, q1);
Rz(q1, -angle/2); // B
CNOT(q0, q1);
Rz(q1, +angle/2); // A
Rz(q0, +angle/2); // D
```

Correct, operation C unneeded

```
Rz(q1, -angle/2);
CNOT(q0, q1);
Rz(q1, +angle/2);
CNOT(q0, q1);
Rz(q0, +angle/2); // D
```

Incorrect, angles flipped

Defense 3: support for modules and unit tests

Bug type 4A: iterating operations

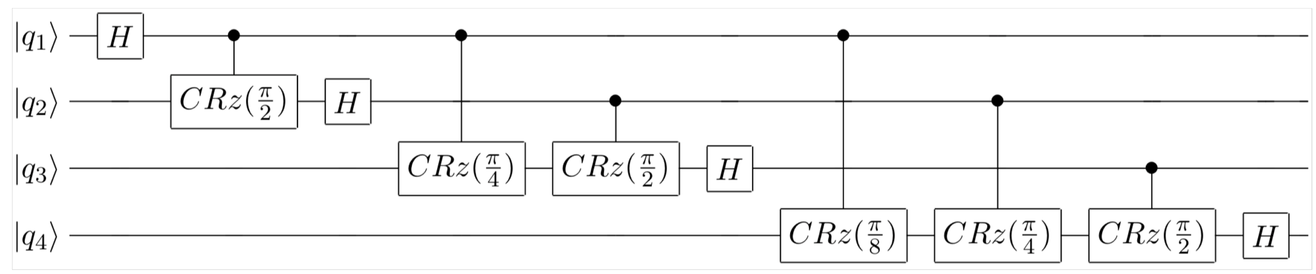


Image credit: Metodi, Faruque, and Chong, Quantum Computing for Computer Architects, 2nd Ed., p26

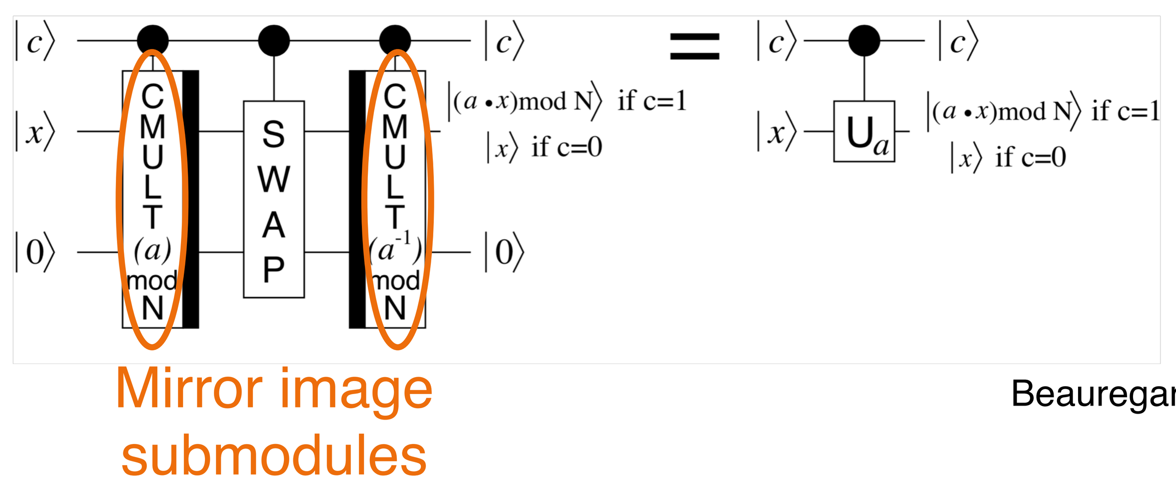
Defense 4A: support for numeric data types

Bug type 4B: recursing operations

```
module cADD (
    const unsigned int c_width, // number of control qubits
    qbit ctrl0, qbit ctrl1, // control qubits
    const unsigned int width, const unsigned int a, qbit b[]
) {
    for (int b_indx=width-1; b_indx>=0; b_indx--) {
        for (int a_indx=b_indx; a_indx>=0; a_indx--) {
            if ((a >> a_indx) & 1) { // shift out bits in constant a
                double angle = M_PI/pow(2,b_indx-a_indx); // rotation angle
                    switch (c_width) {
                        case 0: Rz ( b[b_indx], angle ); break;
                        case 1: cRz ( ctrl0, b[b_indx], angle ); break;
                        case 2: ccRz ( ctrl0, ctrl1, b[b_indx], angle ); break;
}}}}}
```

Defense 4B: support for controlled operations

Bug type 4C: mirroring operations



Mirror image submodules

Beauregard, 2003

Defense 4C: support for reversible compute

Bug type 5: qubit garbage collection

probability		output							
		0	1	2	3	4	5	6	7
ancilla	0	1/8	0	1/8	0	1/8	0	1/8	0
	4	1/64	1/64	1/64	1/64	1/64	1/64	1/64	1/64
	7	1/64	1/64	1/64	1/64	1/64	1/64	1/64	1/64
	8	1/64	1/64	1/64	1/64	1/64	1/64	1/64	1/64
	13	1/64	1/64	1/64	1/64	1/64	1/64	1/64	1/64

Defense 5: post-condition assertions