A systems view of quantum computer engineering

Monday, September 14, 2020

Rutgers University

Yipeng Huang

Due Sunday, Sept. 20: read one of four articles. Share sketches / notes.

- Quantum Computing: Progress and Prospects. National Academies Report. 2019.
- Quantum Computer Systems for Scientific Discovery. NSF. 2019.
- Challenges and Opportunities of Near-Term Quantum Computing Systems. IBM. 2019.
- Quantum computer architecture: towards full-stack quantum accelerators. Delft. 2019.

The role of abstractions in classical computing

What is an abstraction?

What are examples of abstractions?

Why are abstractions good and important?

The role of abstractions in classical computing

What is an abstraction?

What are examples of abstractions?

- APIs, Python, C, assembly, machine code (ISA), CPU-memory (von Neumann), pipeline, gates, binary, discrete time evolution

Why are abstractions good and important?

Hides details so that users & programmers can be creative

The role of abstractions in classical computing

Are abstractions always good?

What are examples of deliberately breaking abstractions?

The role of abstractions in classical computing

Are abstractions always good?

What are examples of deliberately breaking abstractions?

- Python calling C binary (Breaking interpreted high level PL abstraction)
- Assembly code routines (Breaking structured programming abstraction)
- FPGAs (Breaking ISA abstraction)
- ASICs (Breaking von Neumann abstraction)

These two lectures: Broad view of open challenges in quantum computer engineering

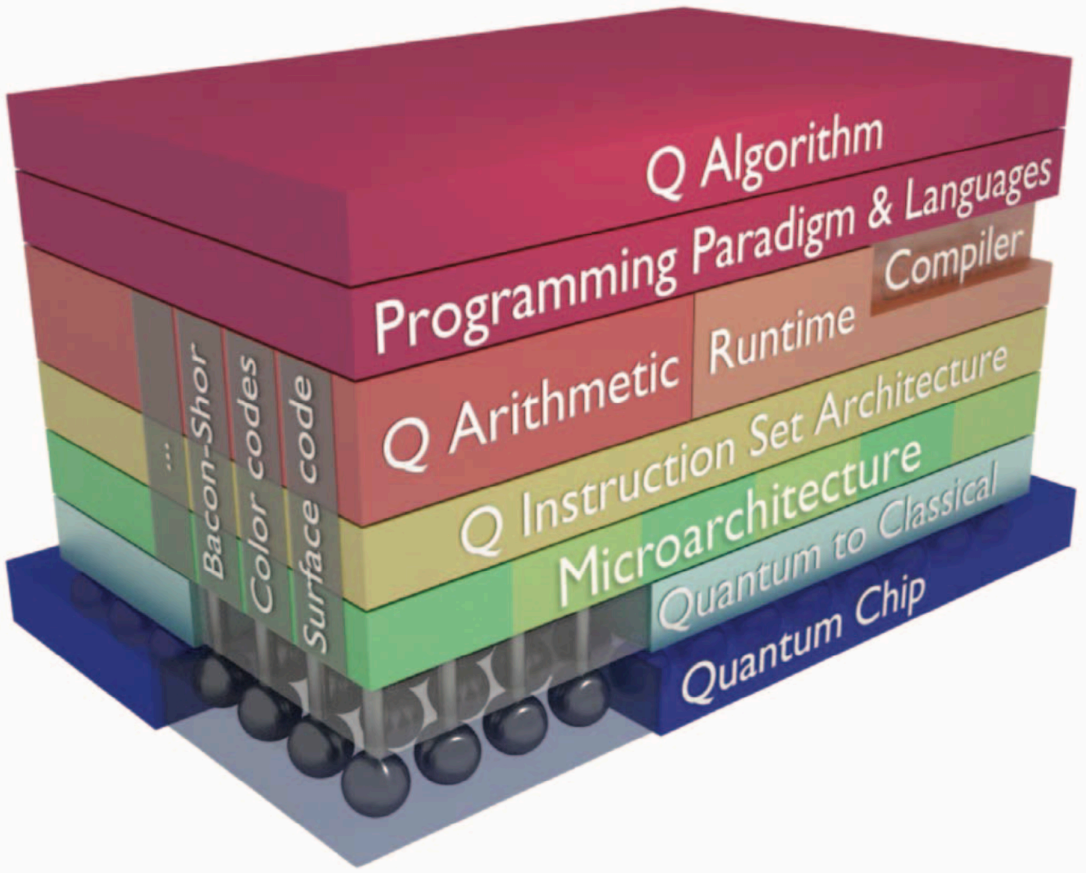


Figure 1. Overview of the quantum computer system stack.

A Microarchitecture for a Superconducting Quantum Processor. Fu et al.

- A complete view of full-stack quantum computing.
- In short, challenges are in finding and building abstractions.
- In each layer, why we don't or can't have good abstractions right now.
- Recent and rapidly developing field of research.

All the quantum computer abstractions we don’t yet have right now

0. Quantum computer support for quantum computer engineering
1. Fault-tolerant, error-corrected quantum algorithms
2. Mature, high level quantum programming languages
3. Universally accepted quantum ISAs
4. Uniform, fully connected quantum device architectures
5. Reliable quantum gates and qubits

All the quantum computer abstractions we don't yet have right now

0. ***Quantum computer support for quantum computer engineering***
1. Fault-tolerant, error-corrected quantum algorithms
2. Mature, high level quantum programming languages
3. Universally accepted quantum ISAs
4. Uniform, fully connected quantum device architectures
5. Reliable quantum gates and qubits

Role of simulation in classical computer engineering

- VirtualBox
- Gem5
- Synopsys / Cadence
- Spice

Warner Brothers Pictures

Why is simulation important?

- "Developing good classical simulations (or even attempting to and failing) would also help clarify the quantum/classical boundary.”
  —Aram Harrow

- Development and debugging of quantum algorithm implementations

Classical simulations of quantum computing

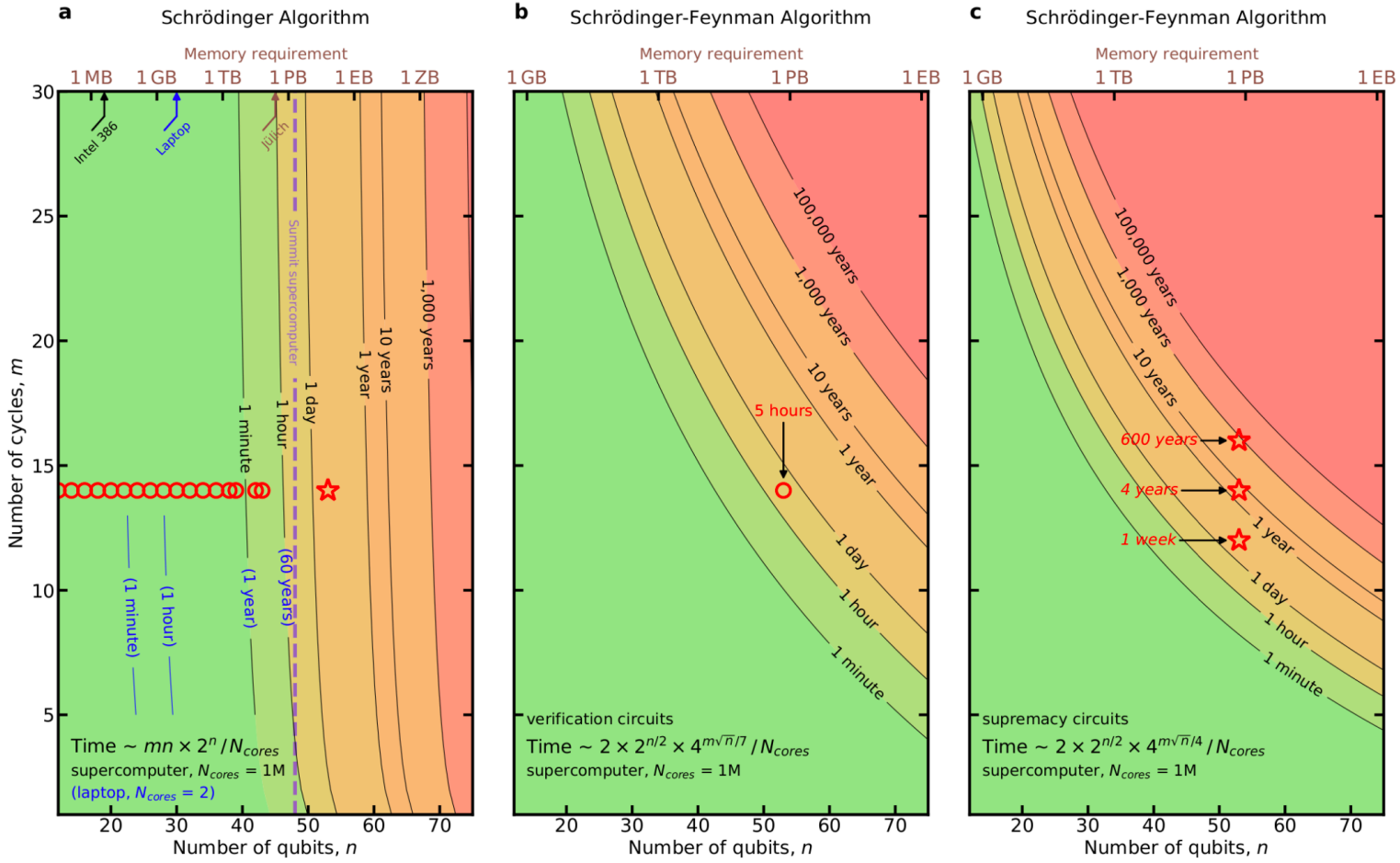


FIG. S40. **Scaling of the computational cost of XEB using SA and SFA. a,** For a Schrödinger algorithm, the limitation is RAM size, shown as vertical dashed line for the Summit supercomputer. Circles indicate full circuits with $n = 12$ to 43 qubits that are benchmarked in Fig. 4a of the main paper. 53 qubits would exceed the RAM of any current supercomputer, and shown as a star. **b,** For the hybrid Schrödinger-Feynman algorithm, which is more memory efficient, the computation time scales exponentially in depth. XEB on full verifiable circuits was done at depth $m = 14$ (circle). **c,** XEB on full supremacy circuits is out of reach within reasonable time resources for $m = 12$, 14, 16 (stars), and beyond. XEB on patch and elided supremacy circuits was done at $m = 14$, 16, 18, and 20.

Quantum supremacy using a programmable superconducting processor (supplement). Arute et al.

- Until we have quantum computer systems, building and testing quantum computers will rely on classical computer systems.

All the quantum computer abstractions we don’t yet have right now

0. Quantum computer support for quantum computer engineering
1. ***Fault-tolerant, error-corrected quantum algorithms***
2. Mature, high level quantum programming languages
3. Universally accepted quantum ISAs
4. Uniform, fully connected quantum device architectures
5. Reliable quantum gates and qubits

Primitives	**Quantum algorithms**
Entanglement protocols	superdense coding / quantum teleportation
Quantum (random) walks	tree traversal
	graph traversal
	satisfiability
Adiabatic	Ising spin model
	quantum approximate optimization algorithm
Variational Quantum Eigensolver	Hamiltonian simulation
Quantum Fourier Transform (QFT)	phase estimation
	period finding
	order finding
	hidden subgroup problem
	linear algebra
Amplitude amplification	database search

QDB: From Quantum Algorithms Towards Correct Quantum Programs. Huang and Martonosi.

Primitives	Quantum algorithms
Entanglement protocols	superdense coding / quantum teleportation
Quantum (random) walks	tree traversal
	graph traversal
	satisfiability
Adiabatic	Ising spin model
	quantum approximate optimization algorithm
Variational Quantum Eigensolver	Hamiltonian simulation
Quantum Fourier Transform (QFT)	phase estimation
	period finding
	order finding
	hidden subgroup problem
	linear algebra
Amplitude amplification	database search

Noisy, intermediate-scale quantum algorithms

Fault tolerant, error corrected quantum algorithms

QDB: From Quantum Algorithms Towards Correct Quantum Programs. Huang and Martonosi.

Grover’s database search algorithm

- Unstructured search problem. Based on amplitude amplification.
- Classical algorithm: needs O(N) queries
- Quantum algorithm: needs O(sqrt(N)) queries

Grover's database search

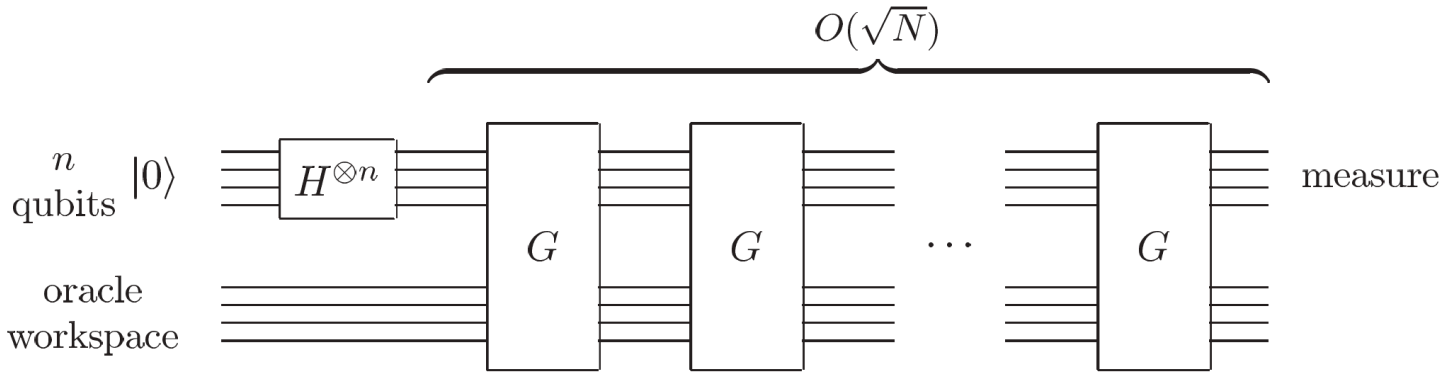


Figure 6.1. Schematic circuit for the quantum search algorithm. The oracle may employ work qubits for its implementation, but the analysis of the quantum search algorithm involves only the n qubit register.

Nielsen and Chuang, p251

Grover's database search

Classical information input

Measurement to obtain classical information

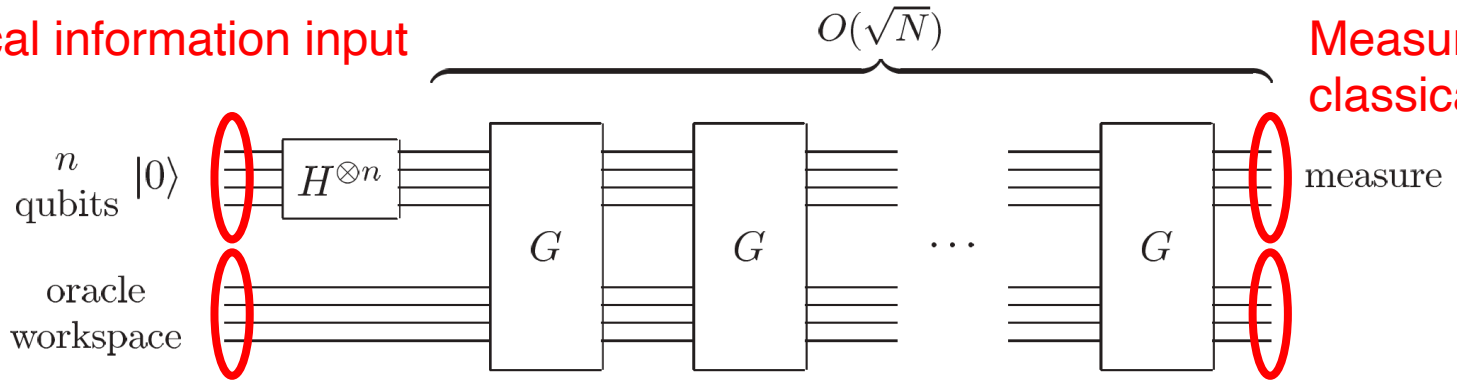


Figure 6.1. Schematic circuit for the quantum search algorithm. The oracle may employ work qubits for its implementation, but the analysis of the quantum search algorithm involves only the n qubit register.

Nielsen and Chuang, p251

Grover's database search

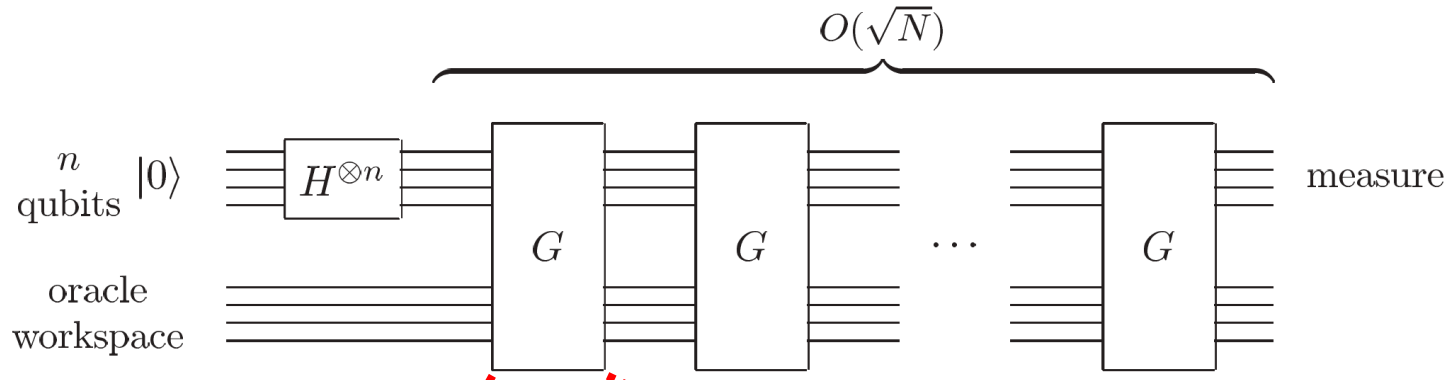
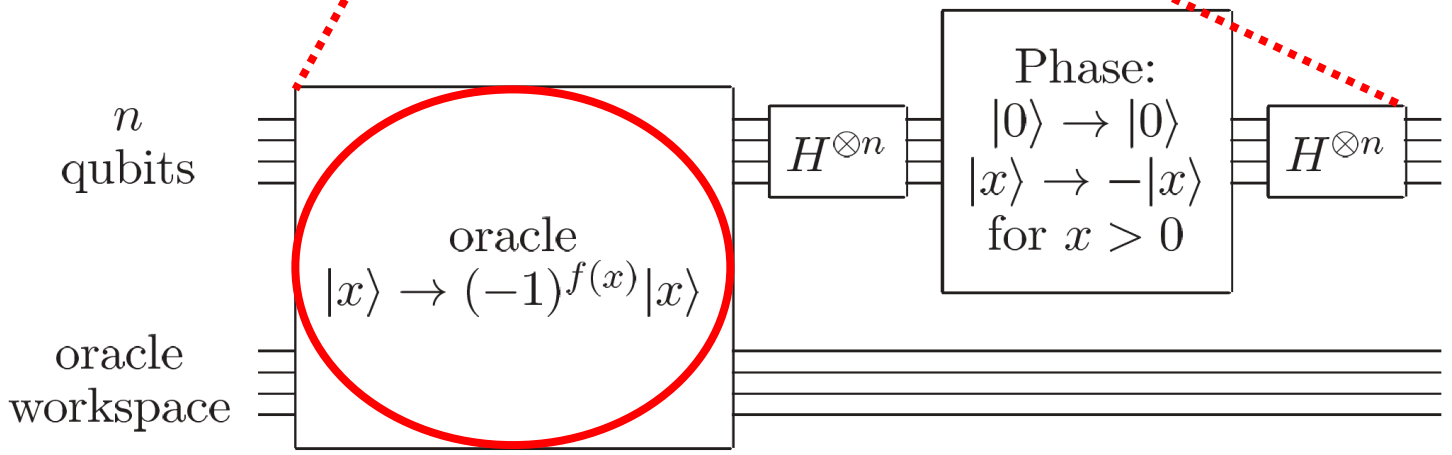


Figure 6.1. Schematic circuit for the quantum search algorithm. The oracle may employ work qubits for its implementation, but the analysis of the quantum search algorithm involves only the n qubit register.



Operations based on classical problem information

Figure 6.2. Circuit for the Grover iteration, G.

Nielsen and Chuang, p251

Grover's database search

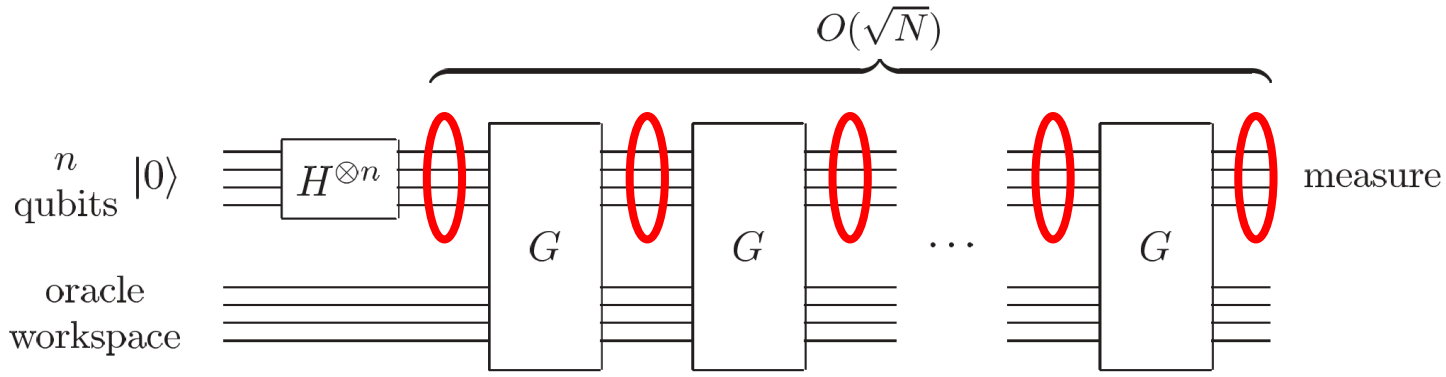


Figure 6.1. Schematic circuit for the quantum search algorithm. The oracle may employ work qubits for its implementation, but the analysis of the quantum search algorithm involves only the n qubit register.

Iterations of quantum circuit operations concentrates the probability amplitude distribution

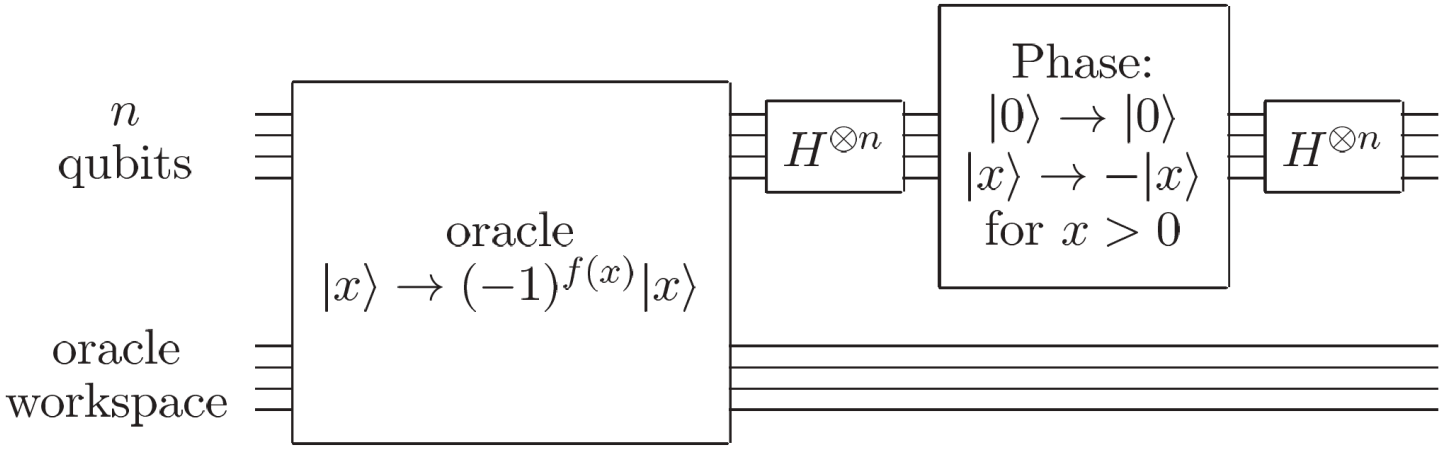
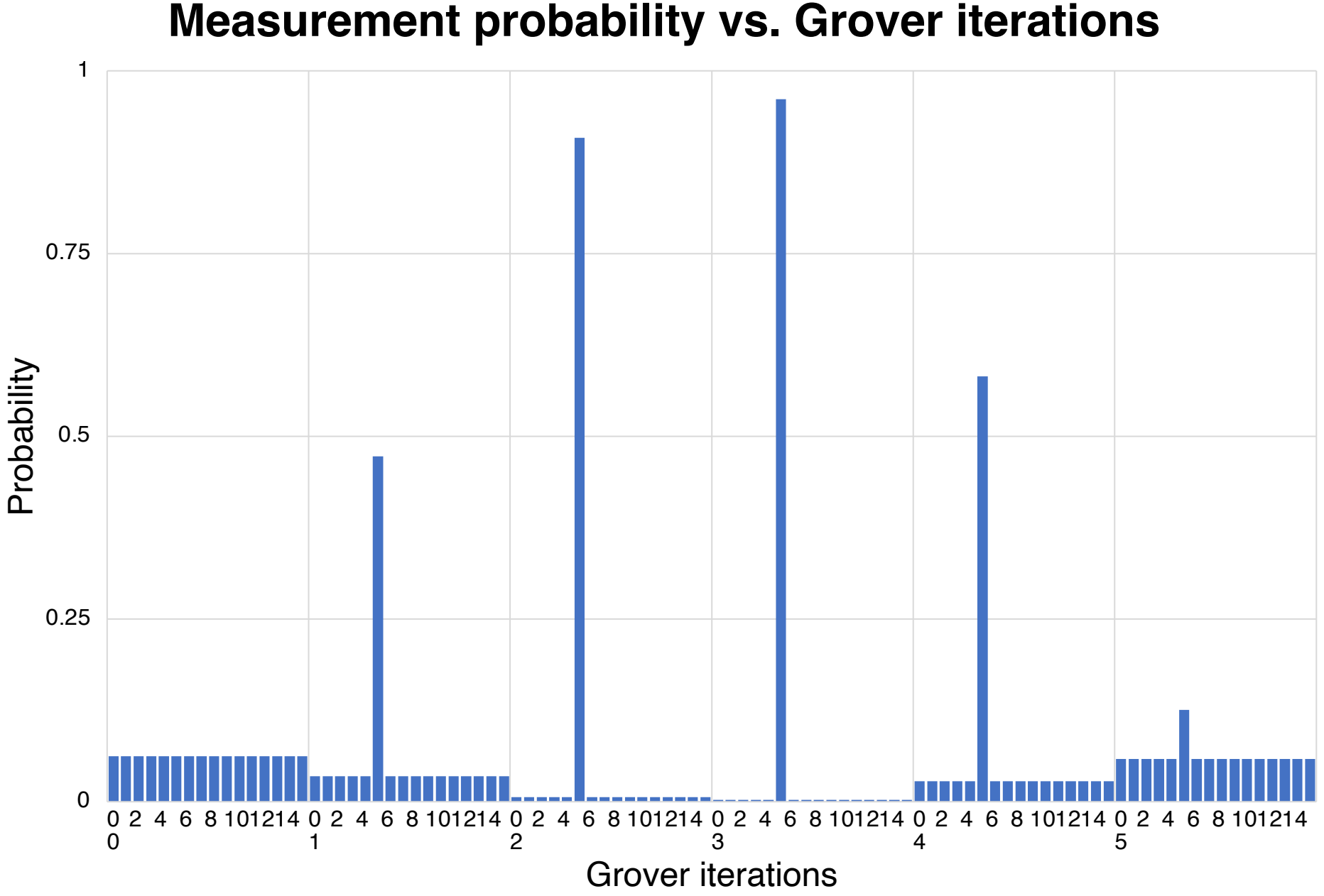


Figure 6.2. Circuit for the Grover iteration, G.

Nielsen and Chuang, p251


Measurement probability vs. Grover iterations
Probability
1
0.75
0.5
0.25
0
0 2 4 6 8 10 12 14
0
0 2 4 6 8 10 12 14
1
0 2 4 6 8 10 12 14
2
0 2 4 6 8 10 12 14
3
0 2 4 6 8 10 12 14
4
0 2 4 6 8 10 12 14
5
Grover iterations

Grover’s database search algorithm

- Unstructured search problem.
- Classical algorithm: needs O(N) queries
- Quantum algorithm: needs O(sqrt(N)) queries

Shor's integer factoring algorithm

Factoring underpins cryptosystems.

For number represented as N bits—

Classical algorithm: needs $O(2^{\sqrt[3]{N}})$ operations

Factoring 512-bit integer: 8400 years. 1024-bit integer: $13*10^{12}$ years.

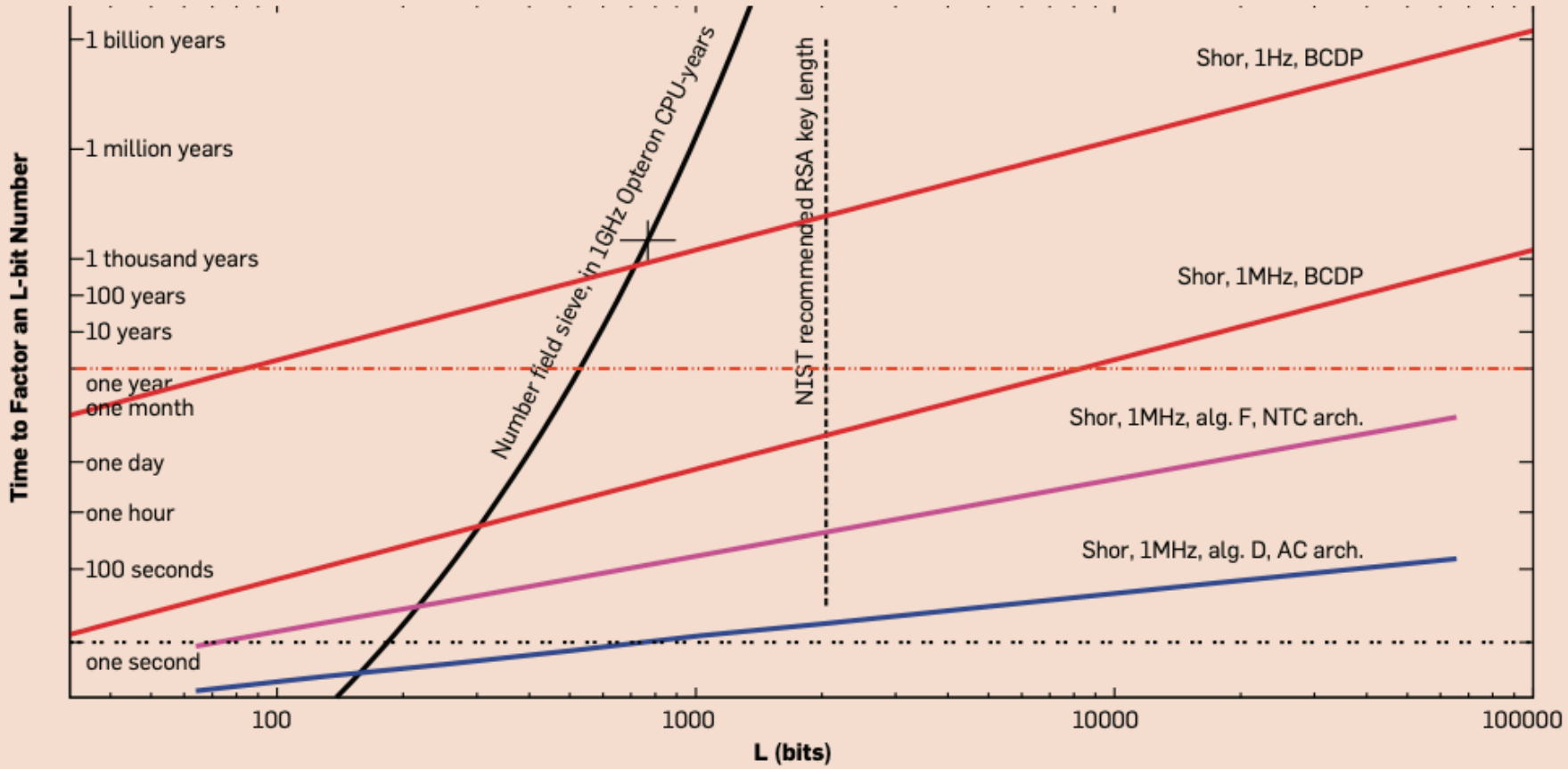
Quantum algorithm: needs $O(N^2 \log(N))$ operations

Factoring 512-bit integer: 3.5 hours. 1024-bit integer: 31 hours.

A Practical Architecture for Reliable Quantum Computers. Oskin et al.

Figure 1. Scaling the classical number field sieve (NFS) vs. Shor's quantum algorithm for factoring.[37]

The horizontal axis is the length of the number to be factored. The steep curve is NFS, with the marked point at $L = 768$ requiring 3,300 CPU-years. The vertical line at $L = 2048$ is NIST's 2007 recommendation for RSA key length for data intended to remain secure until 2030. The other lines are various combinations of quantum computer logical clock speed for a three-qubit operation known as a Toffoli gate (1Hz and 1MHz), method of implementing the arithmetic portion of Shor's algorithm (BCDP, D, and F), and quantum computer architecture (NTC and AC, with the primary difference being whether or not long-distance operations are supported). The assumed capacity of a machine in this graph is $2L^2$ logical qubits. This figure illustrates the difficulty of making pronouncements about the speed of quantum computers.



A Blueprint for Building a Quantum Computer. Van Meter and Horsman.

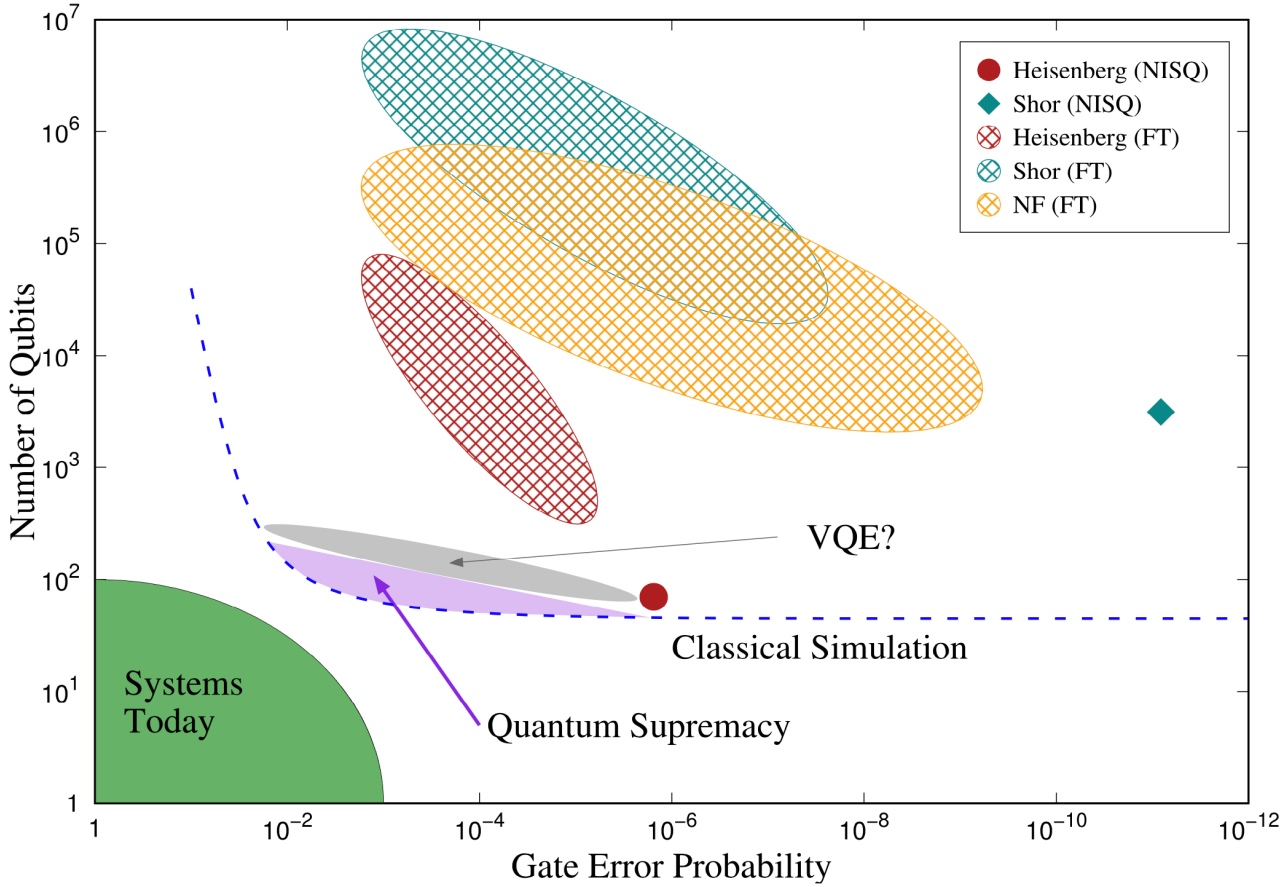


Fig. 2. ***Performance space of quantum computers, measured by the error probability of each entangling gate in the horizontal axis (roughly inversely proportional to the total number of gates that can be executed on a NISQ machine), and the number of qubits in the system in the vertical axis. Blue dotted line approximately demarcates quantum systems that can be simulated using best classical computers, while the green colored region shows where the existing quantum computing systems with verified performance numbers lie (as of September 2018). Purple shaded region indicates computational tasks that accomplish the so-called "quantum supremacy," where the computation carried out by the quantum computer defies classical simulation regardless of its usefulness. The different shapes illustrate resource counts for solving various problems, with solid symbols corresponding to the exact entangling gate counts and number of qubits in NISQ machines, and shaded regions showing approximate gate error requirements and number of qubits for an FT implementation (not pictured are the regions where the error gets too close to the known fault-tolerance thresholds): cyan diamond and shaded region correspond to factoring a 1024-bit number using Shor's algorithm [14], magenta circle and shaded region represent simulation of a 72-spin Heisenberg model [20], and orange shaded region illustrates NF simulation [21].***

An Outlook for Quantum Computing. Maslov et al.

How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. Gidney and Ekerå. 2019.

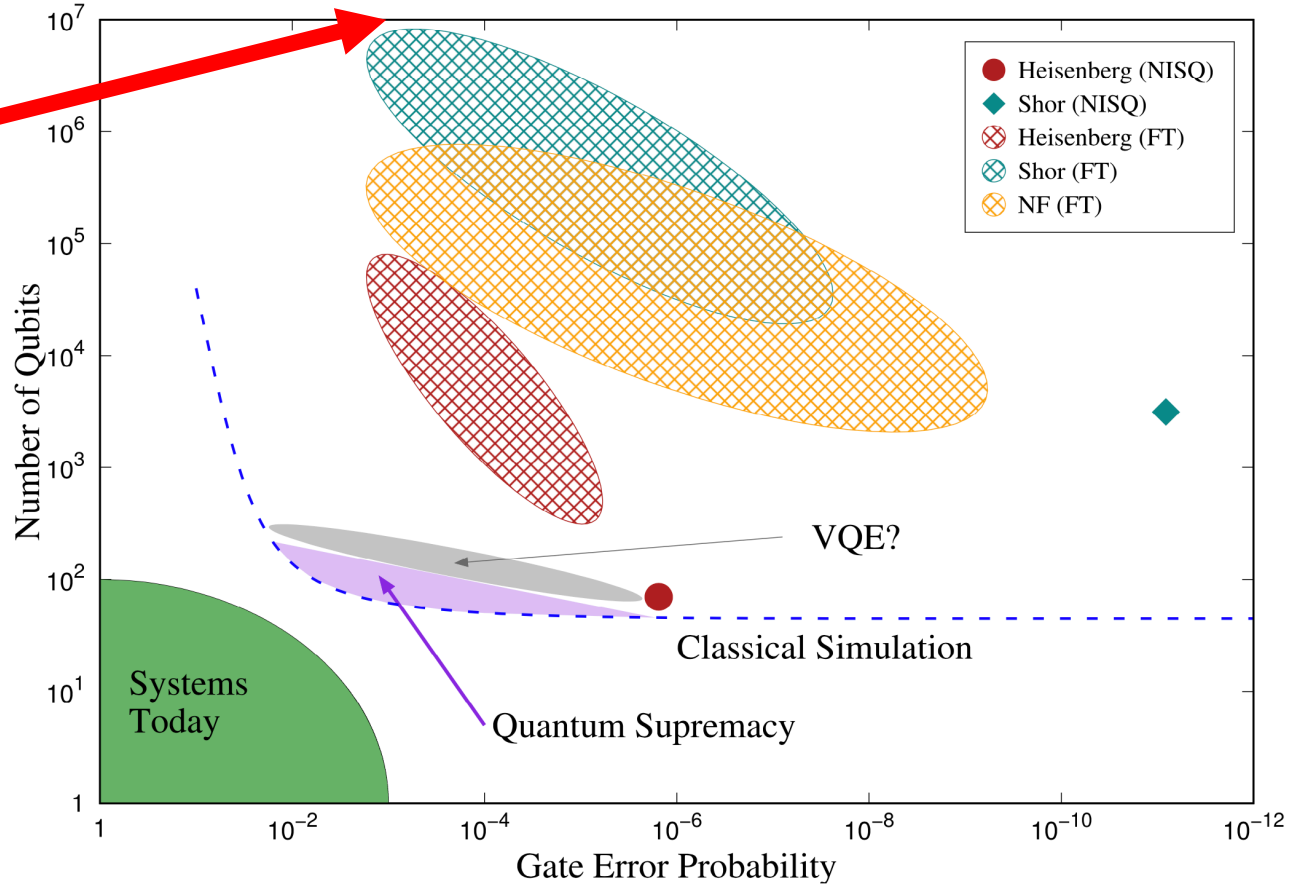


Fig. 2. ***Performance space of quantum computers, measured by the error probability of each entangling gate in the horizontal axis (roughly inversely proportional to the total number of gates that can be executed on a NISQ machine), and the number of qubits in the system in the vertical axis. Blue dotted line approximately demarcates quantum systems that can be simulated using best classical computers, while the green colored region shows where the existing quantum computing systems with verified performance numbers lie (as of September 2018). Purple shaded region indicates computational tasks that accomplish the so-called "quantum supremacy," where the computation carried out by the quantum computer defies classical simulation regardless of its usefulness. The different shapes illustrate resource counts for solving various problems, with solid symbols corresponding to the exact entangling gate counts and number of qubits in NISQ machines, and shaded regions showing approximate gate error requirements and number of qubits for an FT implementation (not pictured are the regions where the error gets too close to the known fault-tolerance thresholds): cyan diamond and shaded region correspond to factoring a 1024-bit number using Shor's algorithm [14], magenta circle and shaded region represent simulation of a 72-spin Heisenberg model [20], and orange shaded region illustrates NF simulation [21].***

An Outlook for Quantum Computing. Maslov et al.

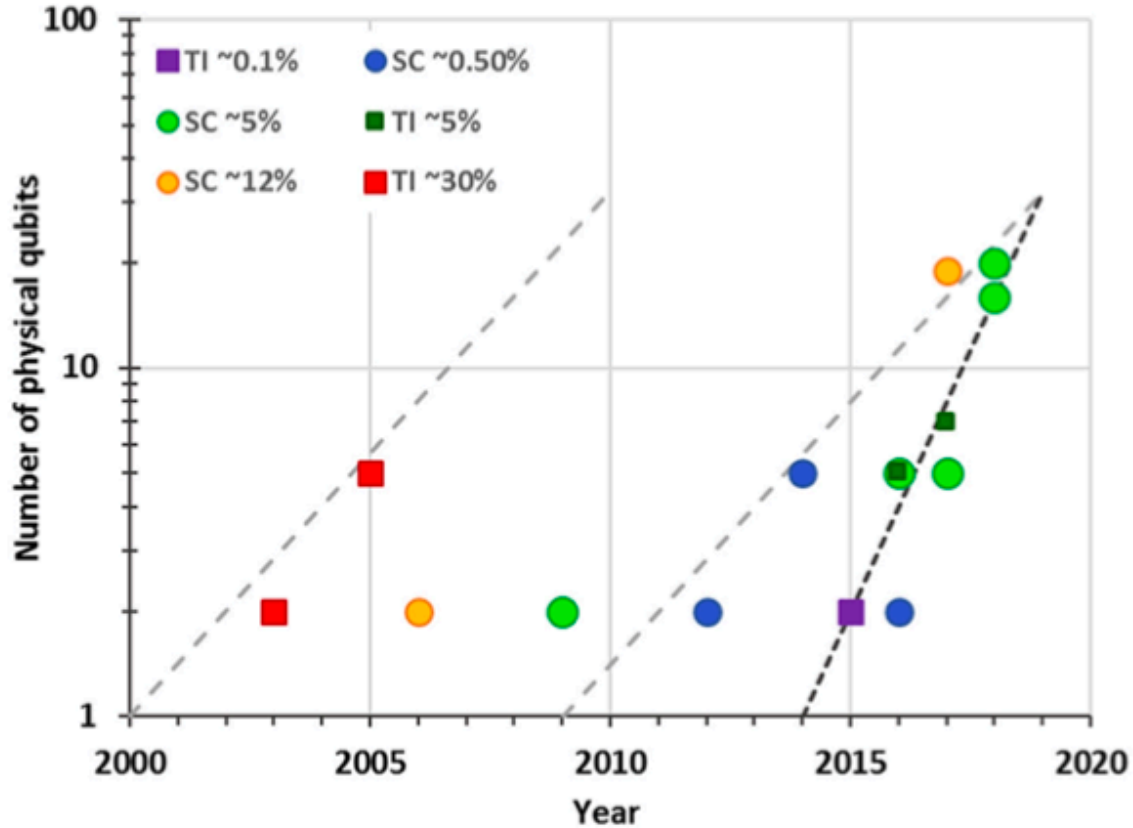
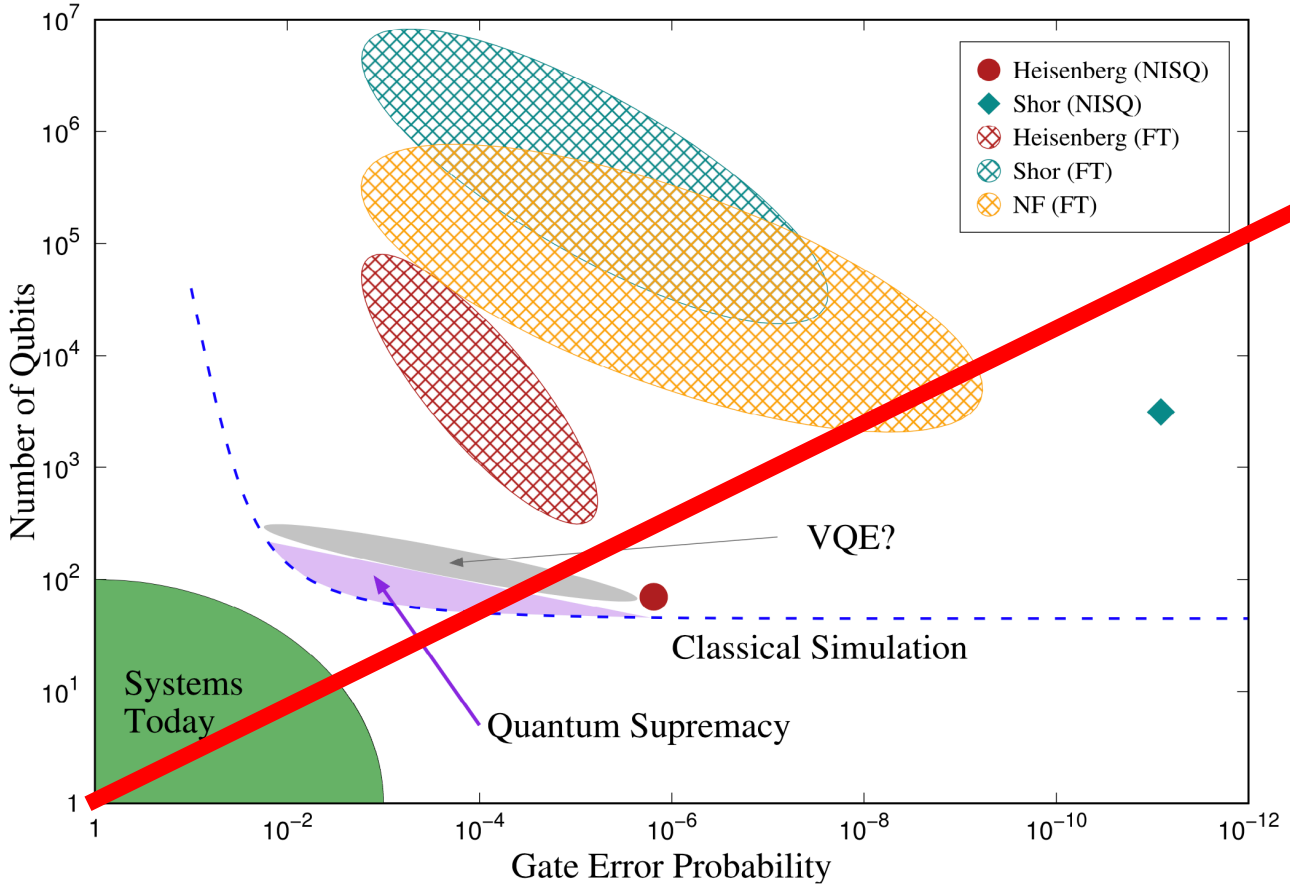


FIGURE 7.2 The number of qubits in superconductor (SC) and trapped ion (TI) quantum computers versus year; note the logarithmic scaling of the vertical axis. Data for trapped ions are shown as squares and for superconducting machines are shown as circles. Approximate average reported two-qubit gate error rates are indicated by color; points with the same color have similar error rates. The dashed gray lines show how the number of qubits would grow if they double every two years starting with one qubit in 2000 and 2009, respectively; the dashed black line indicates a doubling every year beginning with one qubit in 2014. Recent superconductor growth has been close to doubling every year. If this rate continued, 50 qubit machines with less than 5 percent error rates would be reported in 2019. SOURCE: Plotted data obtained from multiple sources [9].

Quantum Computing Progress and Prospects. National Academies Press.



Need applications to motivate further research.

Fig. 2. ***Performance space of quantum computers, measured by the error probability of each entangling gate in the horizontal axis (roughly inversely proportional to the total number of gates that can be executed on a NISQ machine), and the number of qubits in the system in the vertical axis. Blue dotted line approximately demarcates quantum systems that can be simulated using best classical computers, while the green colored region shows where the existing quantum computing systems with verified performance numbers lie (as of September 2018). Purple shaded region indicates computational tasks that accomplish the so-called "quantum supremacy," where the computation carried out by the quantum computer defies classical simulation regardless of its usefulness. The different shapes illustrate resource counts for solving various problems, with solid symbols corresponding to the exact entangling gate counts and number of qubits in NISQ machines, and shaded regions showing approximate gate error requirements and number of qubits for an FT implementation (not pictured are the regions where the error gets too close to the known fault-tolerance thresholds): cyan diamond and shaded region correspond to factoring a 1024-bit number using Shor's algorithm [14], magenta circle and shaded region represent simulation of a 72-spin Heisenberg model [20], and orange shaded region illustrates NF simulation [21].***

An Outlook for Quantum Computing. Maslov et al.

Primitives	Quantum algorithms
Entanglement protocols	superdense coding / quantum teleportation
Quantum (random) walks	tree traversal
	graph traversal
	satisfiability
Adiabatic	Ising spin model
	quantum approximate optimization algorithm
Variational Quantum Eigensolver	Hamiltonian simulation
Quantum Fourier Transform (QFT)	phase estimation
	period finding
	order finding
	hidden subgroup problem
	linear algebra
Amplitude amplification	database search

Noisy, intermediate-scale quantum algorithms

Fault tolerant, error corrected quantum algorithms

QDB: From Quantum Algorithms Towards Correct Quantum Programs. Huang and Martonosi.

All the quantum computer abstractions we don’t yet have right now

0. Quantum computer support for quantum computer engineering
1. Fault-tolerant, error-corrected quantum algorithms
2. ***Mature, high level quantum programming languages***
3. Universally accepted quantum ISAs
4. Uniform, fully connected quantum device architectures
5. Reliable quantum gates and qubits

Table 7 Grover's amplitude amplification subroutine in two languages, showcasing QC-specific language syntax for reversible computation (rows 2 & 6) and controlled operations (rows 3 & 5).

	Scaffold (C syntax) [13]	**ProjectQ (Python syntax) [36]**

1	`int j;`
	`qbit ancilla[n-1]; // scratch register`
	`for(j=0; j<n-1; j++) PrepZ(ancilla[j],0);`
		`# reflection across`
		`# uniform superposition`

2	`// Hadamard on q`
	`for(j=0; j<n; j++) H(q[j]);`
	`// Phase flip on q = 0...0 so invert q`
	`for(j=0; j<n; j++) X(q[j]);`
		`with Compute(eng):`
		`  All(H) | q`
		`  All(X) | q`

3	`// Compute x[n-2] = q[0] and ... and q[n-1]`
	`CCNOT(q[1], q[0], ancilla[0]);`
	`for(j=1; j<n-1; j++)`
	`  CCNOT(ancilla[j-1], q[j+1], ancilla[j]);`
		`with Control(eng, q[0:-1]):`

4	`// Phase flip Z if q=00...0` `cZ(ancilla[n-2], q[n-1]);`	`  Z | q[-1]`

5	`// Undo the local registers`
	`for(j=n-2; j>0; j-)`
	`  CCNOT(ancilla[j-1], q[j+1], ancilla[j]);`
	`CCNOT(q[1], q[0], ancilla[0]);`
		`# ProjectQ automatically`
		`# uncomputes control`

6	`// Restore q`
	`for(j=0; j<n; j++) X(q[j]);`
	`for(j=0; j<n; j++) H(q[j]);`
		`Uncompute(eng)`

QDB: From Quantum Algorithms Towards Correct Quantum Programs. Huang and Martonosi.

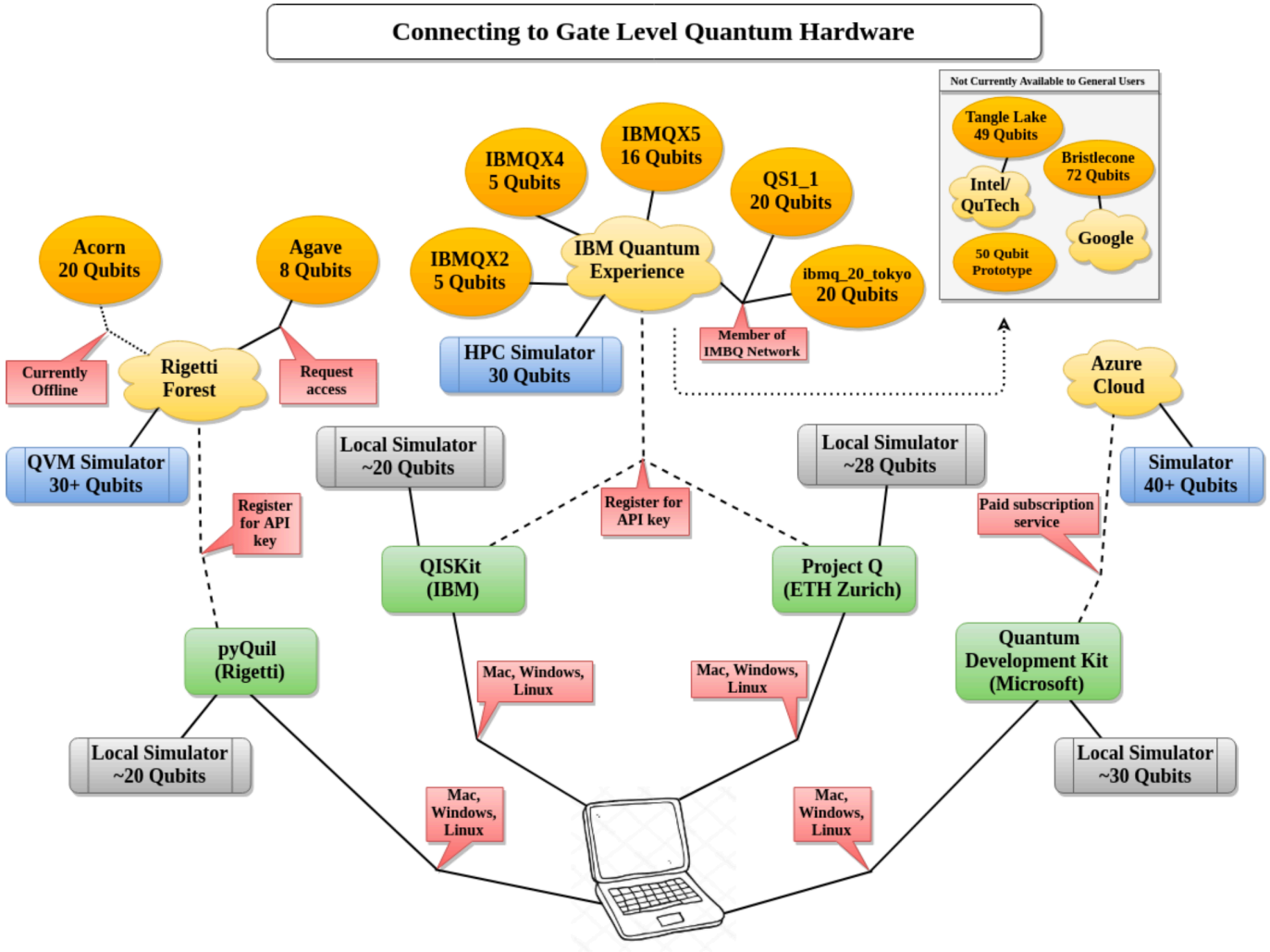


FIG. 1: A schematic diagram showing the paths to connecting a personal computer to a usable gate-level quantum computer. Starting from the personal computer (bottom center), nodes in green shows software that can be installed on the user's personal computer. Grey nodes show simulators run locally (i.e., on the user's computer). Dashed lines show API/cloud connections to company resources shown in yellow clouds. Quantum simulators and usable quantum computers provided by these cloud resources are shown in blue and gold, respectively. Red boxes show requirements along the way. For example, to connect to Rigetti Forest and use the Agave 8 qubit quantum computer, one must download and install pyQuil (available on macOS, Windows, and Linux), register on Rigetti's website to get an API key, then request access to the device via an online form. Notes: (i) Rigetti's Quantum Virtual Machine requires an upgrade for more than 30 qubits, (ii) local simulators depend on the user's computer so numbers given are approximates, and (iii) the grey box shows quantum computers that have been announced but are not currently available to general users.

All the quantum computer abstractions we don’t yet have right now

0. Quantum computer support for quantum computer engineering
1. Fault-tolerant, error-corrected quantum algorithms
2. Mature, high level quantum programming languages
3. Universally accepted quantum ISAs
4. Uniform, fully connected quantum device architectures
5. Reliable quantum gates and qubits