Lecturer: Fred Chong *Scribe: Yongshan Ding*

Lecture 01: Introduction

October 2, 2018

1 Why Quantum Computing

Why focus on quantum computing? Perhaps the largest potential impact of quantum computation is to fundamentally change what is efficiently computable. Quantum computing is the only method by which we can potentially scale computation exponentially with the number of devices. This scaling may allow us to soon solve currently intractable problems in chemistry, simulation, and optimization. As Moore's Law slows, quantum scaling may become an important technique for some key applications, fueling the next generation of computing. Remarkably, research in quantum computing has not only provided insights in other sciences, notably simulations in chemistry, precise control over complex systems in physics, and classical cryptography, but also led to a healthy competition between classical algorithms and quantum algorithms. Progress in quantum algorithms challenges classical algorithms to to better.

Recent progress in quantum hardware has been impressive. IBM is testing a machine with 50 quantum bits (qubits) using superconducting technology, Google has announced their 72-qubit superconducting machine, and Innsbruck has 20-qubit machine based on trapped-ion technology, as of October 2018. Many others in both industry and academia are building similar scale machines in the near future. Machines up to 100 qubits are around the corner, and even 1,000 qubits appears to no longer be a distant dream. John Preskill, a long-time leader in quantum computing at Caltech, notes that we are at a "priviledged time in the history of science and technology" [Pre18]. Specifically, classical supercomputers are not believed to be able to simulate quantum machines larger than 50-100 quantum bits. Emerging physical machines will bring us into unexplored territory and will allow us to learn how real computations scale in practice.

The key to quantum computation is that every additional qubit doubles the potential computing power of a quantum machine. That doubling, however, is not perfect as these machines will have high error rates for some time come. Preskill has a good term to these exciting, but noisy machines: NISQ (Noisy Intermediate-Scale Quantum computers). Ideally, we would use quantum error correction codes to support error-free quantum computations. These codes, however, require many physical qubits to create a single, fault-tolerant qubit; transforming a 100 qubit machine into 3-5 usable logical qubits. Until qubit resources are much larger, a practical approach would be to explore error-tolerant algorithms and use lightweight error-mitigation techniques. So NISQ machines imply living with errors and exploring the effects of noise on the performance and correctness of quantum algorithms.

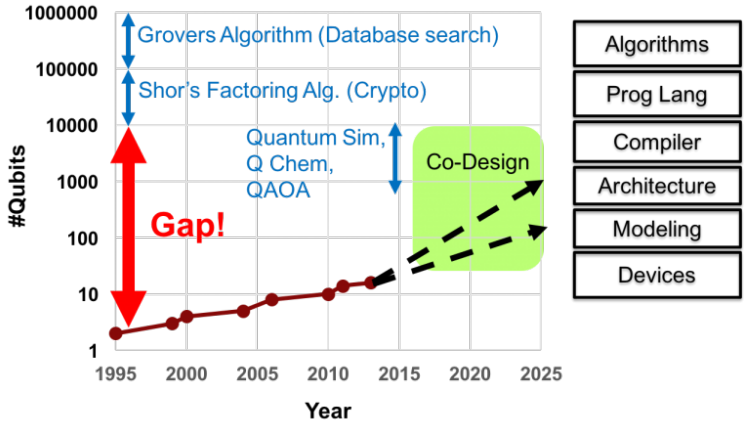

Figure 1: The gap between algorithms and realistic machines.

Additionally, despite technology advances, there remains a wide gap between the machines we expect and the algorithms necessary to make full use of their power. In the Figure 1, we can see the size of physical machines (in this case trapped ion machines) over time. Ground-breaking theoretical work produced Shor's algorithm [Sho99] for the factorization of the product of two primes and Grover's algorithm [Gro96] for quantum search, but both would require machines many orders of magnitude larger than currently practical. This gap has led to a recent focus on smaller-scale, heuristic quantum algorithms in areas such as quantum simulation, quantum chemistry, and quantum approximate optimization (QAOA) [FGG14]. Even these smaller-scale algorithms, however, suffer from a gap of two to three orders of magnitude from recent machines. Relying on solely technology improvements may require 10-20 years to close even this smaller gap.

2 Closing the Gap with Hardware-Software Co-Design

A promising way to close this gap sooner is to create a bridge from algorithms to physical machines with a software-architecture stack that can increase qubit and gate efficiency through automated optimizations and co-design. For example, recent work on quantum circuit compilation tools has shown that automated optimization produces significantly more efficient results than hand-optimized circuits, even when only a handful of qubits are involved. The advantages of automated tools will be even greater as the scale and complexity of quantum programs grows. Other important targets for optimization are mapping and scheduling computations to physical qubit topologies and constraints, specializing reliability and error mitigation for each quantum application, and exploiting machine-specific functionality such as multi-qubit or analog operators.

Although the progress of quantum machines is often measured in terms of qubits, an equally important metric is that of the number of gates (operations) that a machine supports. In fact, the product of qubits times gates would be a more accurate measure of the capabilities

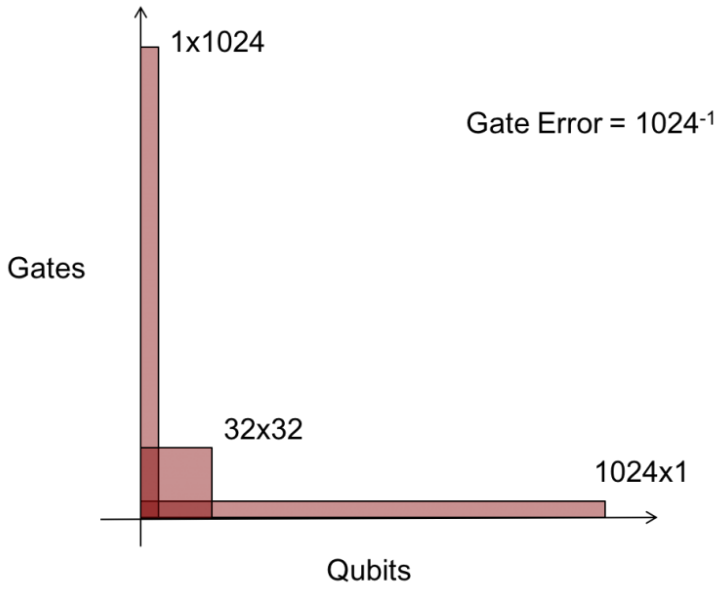

Figure 2: Space-time (quantum volume) constraints on machines due to gate errors.

of a machine, as shown in Figure 2. For example, if we have gate error of 1 in 1024, that means that we can have a 1-qubit machine that can perform 1024 gates on that qubit before expecting an error. Similarly, we could have 1024 qubits and only perform a single gate. Finally, we could have a more balanced system with 32 qubits, each of which could undergo 32 gate operations. Other details include that 2-qubit gates are more difficult than 1-qubit gates, and that the interconnect between each physical qubit limits the effectiveness of a machine. IBM refers to the effective scaling a quantum machine as its quantum volume [BBC+17]. Optimistically, we might hope to see a machine with a quantum volume of 100 qubits times 1,000 gates in the next couple years. So more precisely, the gap between algorithms and machines can be measured in terms of this space-time product and increasing the efficiency of algorithms on machines involves reducing both the number of qubits and gates needed.

2.1 Algorithm Design

So what makes a good quantum algorithm o a good problem to solve? Firstly, problems that quantum computers are good at solving in general have very compact input-output representations. This is because data encoded in qubits can only be accessed via measurement - a process that results in a probabilistic readout and is relatively error-prone. For instance, Shor's algorithm solves the prime factorization problem whose input is simply the number we are factoring and the output is its prime factors, which have compact bit representations. Similarly, many molecule simulations and graph optimization algorithms have relatively small input-output representations but large amount of computation in between.

Therefore, quantum big data is probably not going to be realistic very soon (not until new ways of efficiently solving the I/O problem are invented), due to the significant work that one has to put into encoding and decoding classical data to and from the quantum process.

Secondly, the results of the problems should ideally be easily verifiable. As mentioned, information extracted from a quantum algorithm via measurements are probabilistic and error-prone. So, we typically need multiple trial-and-error style executions so as to amplify the desired outcome. Therefore, efficient ways of verifying the results may help us identify the correct results quicker. For example, the result of Shor's algorithm can be easily verified by multiplying the factors in polynomial time.

Thirdly, while quantum computers are small and unreliable, a great way to exploit their special, but limited, abilities is to adopt a hybrid model which leverages both quantum and classical computation. Perhaps the most promising example is in quantum chemistry, where Variational Quantum Eigensolver (VQE) algorithms [OBK+16] perform a kind of heuristic search by iterating between a quantum machine and a classical supercomputer. The goal is to find the lowest energy state of a chemical compound (the ground state). We start from the best known configuration of electrons from a classical supercomputer and estimate the energy of that configuration using the quantum machine. This estimate is then given back to a classical supercomputer to guide its search towards a configuration with lower energy. In this way, the quantum machine acts as accelerator for the energy modeling part of the computation.

2.2 Computer System Design

2.2.1 Compiler Optimizations

Quantum program compilation now is just like classical computer compilation in the 1950s, where every bit and every gate matter. Due to the scarce and noisy resources given in a quantum computer, it is important to optimize the programs for maximum parallelism, minimal communication overheads, and highest precision. This multi-objective nature makes a quantum program very difficult to optimize by hand. We should therefore approach this not only by adopting traditional compiler techniques [HPJ+15] such as loop-unrolling, constant propagation, inlining, function cloning, and DAG scheduling, but also by developing novel techniques that take advantages of, for example, the structures in quantum algorithms and the commutation relations in quantum circuits.

2.2.2 Architecture

One of the challenges in building a scalable quantum computer is to make the qubits live longer before decoherence happens. Qubits in a NISQ computer are relatively short-lived and gates are relatively noisy, so one big problem to solve in quantum computer architecture is to make sure the classical controls can keep up with the quantum processor. Recently, researchers from TU Delft, The Netherlands have demonstrated an architecture design [FRB+17] that generates operation pulses fast enough to prevent qubit waiting time

by keeping the event queues full and playing back from the queue according to synchronization controls. To do so, they defined an Instruction Set Architecture (ISA). As a result, it provides a clean library of only a few instructions, much like classical computer systems that we are familiar with. However, this level of abstraction is probably leaving too much efficiency on the table. If one stands outside of the box of classical computer architectures, and rethinks about the quantum instructions from a more fundamental point of view, one would argue that quantum instructions, implemented as pulse sequences, in the compiled quantum assembly may not be those that a machine is best at. In other words, suppose one function (possibly on multiple qubits) is comprised of ten assembly instructions; the corresponding ten pulse sequences may not be the best way to achieve this function. Instead, we can find the optimal pulse that take the qubits system from its initial state to the final state with higher precision and faster speed. Our preliminary results show that multi-qubit optimal control pulses are able to achieve an average of 2x and up to 10x improvement on pulse durations.

2.2.3 Memory Management

Quantum applications usually requires a large number of temporary, scratch qubits, called ancillae. This is partly because quantum gates are coherent processes that conserve energy and must be reversible. In classical computing, we can build any logic gate in a circuit out of NAND gates. However, NAND gate is not reversible, as there is no way to reconstruct the two input bits from the single output bit. A reversible gate must have equal number of input and output bits [1]. For example, arithmetics in many quantum applications therefore involve extensive use of ancillae. So how can we recycle them for multiple uses? It turns out that those ancillary qubits cannot be directly reused, because they are sometimes "coupled" with the result qubits and reusing them will corrupt those results. Instead, if one intend to recycle the ancillae for future use, one may need to perform the "uncomputation" step that reverses the coupling and restore the ancillae to their initial state. This will allow us to save the total number of qubits, at the cost of roughly doubling the number of gates. It exposes a direct resource tradeoff in space-time volume.

Additionally, there is an extra twist in memory management, that is locality matters in most quantum devices. Specifically, most quantum machines support only nearest-neighbor interactions between two qubits, e.g. a 2-D mesh of qubits. Therefore, it is crucial that, when one recycles a qubit, it is later reused at a location that is not too far from its original position. This gives rise to a mapping problem where the objective is to minimizing the communication overhead.

There are a number of techniques that we can use to ensure low communication overhead. They generally fall into two categories: one that utilizes global knowledge and the other that relies on local heuristics. The objective is to keep interacting qubits close together. With global knowledge of the interaction graph, we can use algorithms such as recursive graph partitioning that works well if the interaction graph dose not have too many crossings, and clustering algorithm that helps keeping densely connected components local [DHJA+18]. We

[1] This is a necessary but not sufficient condition.

can also adopt greedy approach to solve this problem. At any point in the program, we can determine what's the best mapping at that moment, and move on to the next part in the program from there. This approach is more scalable and flexible, but its solution is often less optimal than that of the global approaches.

2.2.4 Verifying Quantum Programs

In order to get the most out of scarce quantum resources, researchers will need to "cut corners" as much as possible without compromising correctness. This leads to a strong need for a methodology for verifying the correctness of both quantum software and hardware. Coming up with such a methodology is a bit of a grand research challenge, because we do not yet have reliable quantum hardware and it is exponentially-difficult to simulate quantum computations on classical computers (since a quantum computer's state space is exponential in the number of its qubits).

We can either adopt testing-based or formal-methods approach to these verification problems (or hybrids of both). Assuming that we will use classical computation for verification, both approaches skirt a fundamental tension between classical simulation of quantum computations and quantum supremacy. If we can efficiently simulate quantum computation on a classical computer, then we have proven that this quantum computation does not demonstrate quantum supremacy! Verification approaches involving too much of an algorithms state space also tread in a similar space. If we are optimistic and assume that some quantum algorithms have supremacy over classical algorithms, then we must come up with restricted verification properties that only require partial simulation or formal verification of a sub-exponential state space.

Some results from partial simulation illustrate the subtlety of this boundary between classical simulation and quantum supremacy. Classical computers can simulate quantum algorithms consisting of only "Clifford gates" in time polynomial in the number of qubits used in the algorithm, which proves that these algorithms do not demonstrate quantum supremacy. Algorithms such as Shor's factoring algorithm, which are exponentially better than known classical algorithms, contain "T gate" (rotations on quantum vectors) as well as Clifford gates. We do not know how to classically simulate Shor's algorithm in sub-exponential time. We do know, however, how to simulate algorithms consisting of Clifford gates and a very small number of T gates in polynomial time. So the boundary between quantum supremacy and no supremacy is somewhere between a small number of T gates and a lot of T gates in an algorithm. Verification through simulation might exploit the polynomial side of this boundary by trying to define correctness properties that only require simulation of part of an algorithm that contains a small number of T gates. What these properties will be, however, is very much an open area of research.

If we have a simulator or working machine, we can perform end-to-end unit tests or we can invest some extra quantum bits to test assertions. Methods have been developed to test for basic properties such as whether two quantum states are equal, whether two states are entangled, or whether operations commute. Some progress has also been made in applying formal methods to verify quantum computations. QWire [RPZ18] uses coq to verify some

properties of simple quantum circuits, but classical computation for the theorem prover scales exponentially with the number of qubits. Once again, the key challenge is to define useful correctness properties that a theorem prover can handle more scalably.

References

[BBC+17] Lev S Bishop, Sergey Bravyi, Andrew Cross, Jay M Gambetta, and John Smolin. Quantum volume. Technical report, Technical report, 2017. URL: https://dal. objectstorage. open. softlayer. com/v1/AUTH_039c3bf6e6e54d76b8e66152e2f87877/community-documents/quatnum-volumehp08co1vbo0cc8fr. pdf, 2017.

[DHJA+18] Yongshan Ding, Adam Holmes, Ali Javadi-Abhari, Diana Franklin, Margaret Martonosi, and Frederic T Chong. Magic-state functional units: Mapping and scheduling multi-level distillation circuits for fault-tolerant quantum architectures. *arXiv preprint arXiv:1809.01302*, 2018.

[FGG14] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[FRB+17] Xiang Fu, MA Rol, CC Bultink, J Van Someren, Nader Khammassi, Imran Ashraf, RFL Vermeulen, JC De Sterke, WJ Vlothuizen, RN Schouten, et al. An experimental microarchitecture for a superconducting quantum processor. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 813–825. ACM, 2017.

[Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[HPJ+15] Jeff Heckey, Shruti Patil, Ali JavadiAbhari, Adam Holmes, Daniel Kudrow, Kenneth R Brown, Diana Franklin, Frederic T Chong, and Margaret Martonosi. Compiler management of communication and parallelism for quantum computation. *ACM SIGARCH Computer Architecture News*, 43(1):445–456, 2015.

[OBK+16] PJJ OMalley, Ryan Babbush, ID Kivlichan, Jonathan Romero, JR McClean, Rami Barends, Julian Kelly, Pedram Roushan, Andrew Tranter, Nan Ding, et al. Scalable quantum simulation of molecular energies. *Physical Review X*, 6(3):031007, 2016.

[Pre18] John Preskill. Quantum computing in the nisq era and beyond. *arXiv preprint arXiv:1801.00862*, 2018.

[RPZ18] Robert Rand, Jennifer Paykin, and Steve Zdancewic. Qwire practice: Formal verification of quantum circuits in coq. *arXiv preprint arXiv:1803.00699*, 2018.

[Sho99] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.