Off-the-shelf Components for Quantum Programming and Testing

Cláudio Gomes[a], Daniel Fortunato[c], João Paulo Fernandes[a] and Rui Abreu[b]

[a]CISUC — Departamento de Engenharia Informática da Universidade de Coimbra, Portugal
[b]Faculty of Engineering of the University of Porto, Portugal
[c]Instituto Superior Técnico, University of Lisbon, Portugal

Abstract

In this position paper, we argue that readily available components are much needed as central contributions towards not only enlarging the community of quantum computer programmers, but also in order to increase their efficiency and effectiveness. We describe the work we intend to do towards providing such components, namely by developing and making available libraries of quantum algorithms and data structures, and libraries for testing quantum programs. We finally argue that Quantum Computer Programming is such an effervescent area that synchronization efforts and combined strategies within the community are demanded to shorten the time frame until quantum advantage is observed and can be explored in practice.

Keywords

Quantum Computing, Software Engineering, Reusable Components

1. Introduction

There is a large body of compelling evidence that *Computation* as we have known and used for decades is under challenge. As new models for computation emerge, its limits are being pushed beyond what pragmatically had been seen in practice. In this line, Quantum Computing (QC) has received renewed worldwide attention. Having its foundations been thoroughly studied, mainly from the point of view of its physical implementation, their potential has, even if preliminarily, is currently being witnessed.

A quantum computer can potentially solve various problems that a classical computer cannot solve efficiently; this is known as Quantum Supremacy. Examples include scalable simulations of quantum systems in physics, efficient modelling of chemical reactions, and fast breaking of encryption codes in cryptography.

In an article published in Nature in October 2019, Google describes how using a self-built 54-qubit processor correctly executed, in only 200 seconds, a benchmark that even the world's

Q-SET'20: 1st Quantum Software Engineering and Technology Workshop, October 13, 2020, Denver — Broomfield, Colorado, USA
EMAIL: gomes@student.dei.uc.pt (C. Gomes); daniel.b.fortunato@tecnico.ulisboa.pt (D. Fortunato); jpf@dei.uc.pt (J.P. Fernandes); rui@computer.org (R. Abreu)
URL: http://dei.uc.pt/~jpf (J.P. Fernandes); https://ruimaranhao.com/ (R. Abreu)
ORCID: 0000-0002-1952-9460 (J.P. Fernandes); 0000-0003-3734-3157 (R. Abreu)

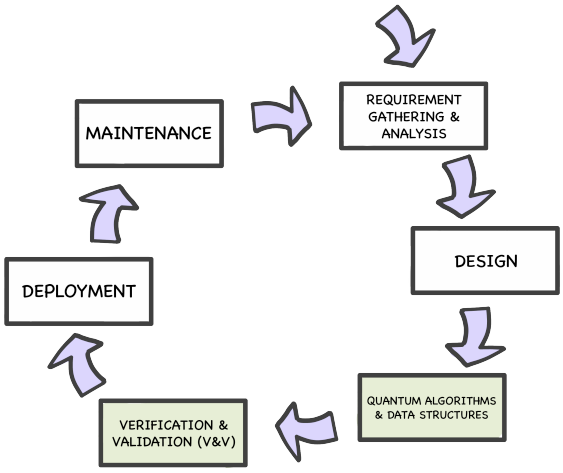

Figure 1: Software Development life-cycle (colored boxes are the focus of this position paper).

fastest supercomputer would have taken an estimation of 10,000 years to complete; this way showing the so-called quantum supremacy.

In a follow-up, IBM has disputed the foundations of such estimation, and mainly the claim that quantum supremacy has been reached. IBM's argument is mainly about the assertion that a properly crafted supercomputer could have reached the same result even more efficiently than the Google quantum computer. However, no empirical demonstration was provided to support such assertion. In essence, Google's experiment provides clear evidence of the progress that has been made in terms of superconducting-based quantum computing. IBM itself has also made substantial progress to build universal quantum computers to support business, engineering and science.

The field of QC is evolving at a pace faster than people originally expected. For example, in September 2020 Honeywell announced a that its revolutionary quantum computer based on trapped-ion technology with achieved a quantum volume 128 – the highest quantum volume ever achieved, and twice as the previous state of the art. Quantum volume is a unit of measure indicating the fidelity of a quantum system. This important achievement shows that the field of quantum computing may reach industrial impact much sooner than originally expected.

While the fast approaching universal access to quantum computers is bound to break several computation limitations that have lasted for decades, it is also poses major challenges in many, if not all, computer science disciplines. It is well known, e.g., that the foundations of modern cryptography based on the prime number factorization problem will have to be reconsidered.

In this position paper, we propose to explore the potential and study the implications of QC under the lenses of Software Engineering, which entail several phases during for the development life-cycle (see Figure 1). Despite there is the need to also advance the state-of-the-art of the other phases, we propose to focus on the phases of implementation and validation of quantum programs, both when considered in isolation, and in a hybrid approach that combines quantum and classical programs. We argue that these two phases are the much needed work to make quantum programming accessible to people outside the quantum mechanics world.

Our initial goal, which is described in Section 2, is to propose abstraction mechanisms to improve the state of the art in terms of quantum software implementation. As the current approaches to quantum programming also resort to quantum gates, they require a significant effort from programmers. We will provide more efficient development mechanisms by implementing a library of (hybrid) algorithms and data structures whose classical implementation is well known and widely used. This library will be published as an open source artifact that the community can build upon.

Furthermore, approaches to perform verification and validation of quantum programs are essentially lacking and largely unexplored. In fact, having implemented a quantum program, the current practice to try to establish its correctness is to run the program multiple times and observe its probable result. Although programmers can already run a program on a quantum computer, there is no abstraction layer to make testing, let alone verification or validation, more effective. As described in Section 3 We will propose black-box methods to efficiently test programs running on a quantum computer. Black-box methods are especially suited because, due to the underlying classic quantum mechanics, one cannot observe the inner workings of a quantum program without altering the program's state and the final result, as measuring a qubit destroys superposition.

2. Quantum algorithms and data structures

In 1976, in the title of his landmark textbook[1], Niklaus Wirth coined a famous equation in Computer Science: $Algorithms + DataStructures = Programs$.

The sharpness of the equation supports the argument that mastering programming can not be achieved without the combined knowledge of both algorithms and data structures.

When defining a data structure, one's goal is to represent an entity from the real world as a string of bits in such a way that queries about that entity can be established efficiently. Studying data structures aims precisely at finding the sweet spot between the length of the bit string and the time it takes to answer queries on it.

Problems associated with data structures are often divided into two categories: static and dynamic. For static problems, we are essentially interested in being able to answer queries over a data structure. For dynamic problems, we additionally want to be able to efficiently change the data structure content.

We propose to study and implement data structures in quantum processing units both for static and dynamic problems. A concrete research challenge that the quantum context entails is that query operations typically need to inspect, or measure, the (qu)bit string, which may irreversibly alter the corresponding data. This differs from the classical context in the sense that now we will also need to consider how many times one can use a data structure.

We will start by targeting classical data structures with quantum access before moving on to explore fully quantum data structures. In the former model, data is stored in classical bits, which can be accessed quantumly. While this approach is certainly constructive, it has been shown that for most problems, it has no (asymptotical) advantage over classical data structures. We will then move on to the fully quantum realm, where data is encoded in qubits instead of bits, and where we have access to all the operations that are provided in quantum computing.

We will also considered extensively and in depth the quantum-setting algorithmic counterpart of Wirth's equation.

Quantum computation models have originally been studied in the 1980s and algorithms to explore quantum computing started appearing in the early 1990s, even if back then quantum computational devices were essentially theoretical.

Two of the most well known quantum algorithms are Shor's polynomial-time algorithm for prime factorization and discrete logarithm and Grover's algorithm that can efficiently search data in an unstructured database.

While the groundbreaking nature of these well known algorithms can not be denied, a challenge that we face here is that they are not possible to be executed in the near-term. While near-term hardware implementations (of less than a hundred qubits) have recently been developed, their limitations pose significant challenges regarding the development of practical algorithms, the ones we propose to target. Nevertheless, a number of NISQ algorithms have already been proposed, namely the Variational Quantum Eigensolver[2] and the Quantum Approximate Optimization Algorithm [3], which demonstrate the feasibility of the approach.

The development of quantum algorithms is particularly challenging since it requires a combination of skills which include, e.g., quantum mechanics and complexity theory, as well as a deep computer science background. This will be strategically considered by representing different profiles to supervise and conduct the associated workplan.

The outcomes of our work will be made publicly available as open source tool sets that can benefit the entire community. These artifacts will target multiple platforms that are already available for general-purpose quantum computing such as IBM Qiskit or Google's Cirq.

3. Validation and verification (V&V) of quantum programs

Quantum programming is challenging as quantum programs are necessarily probabilistic and impossible to examine without disrupting execution. This difficulty is compounded by the fact that quantum programs are difficult to test and/or debug.

In the Classical Computing realm, V&V has been extensively addressed. Classical computing V&V techniques, however, cannot be applied off-the-shelf.

Consider, e.g., two standard techniques for debugging programs: breakpoints and print statements. In a quantum program, printing the value of a quantum bit entails measuring it and printing the returned value (i.e., measuring a qubit), which destroys superposition. Unit tests are similarly of limited value when a program is probabilistic; repeatedly brute-forcing/running unit tests on a quantum computer may be prohibitively expensive. Simulating quantum programs on a classical computer holds some promise but requires resources exponential in the number of qubits, so simulation cannot help in the general case.

In the context of Quantum Computing, V&V is still in its infancy[4]. A venue that has been explored is to reason about a quantum program with vectors, as vectors do not collapse when being analyzed. As formal verification is parametric in its inputs, one can prove properties of that algorithm for arbitrary arities given as arguments. Preliminary advances have been shown in a recent work using techniques like induction and algebraic reasoning[5].

We argue that, in part, the lack of study in the field may be attributed to the fact that

programming languages available for QC are still essentially low-level, operating at the level of quantum gates [6, 5]. However, Quantum Programming languages are emerging [7, 8]. While high-level constructs aim at improving productivity, they also aim to allow programmers to express even more complex computations.

In this context, we aim to devise novel optimized quantum testing techniques to help develop high quality quantum programs, thus leading to more confidence when deploying the envisioned future safety and mission-critical applications of quantum computing in the long-term. Thus, we will attempt to build theoretical foundations complemented with technologies for testing quantum programs and, therefore, ensuring the dependability of quantum applications.

Another challenge of addressing V&V within Quantum Computing is that the systems of the future will be hybrid, in the sense that the majority of software features will still be implemented 'classically', while computation-intensive features may be delegated to quantum components. This means that our studies need to combine strategies that have demonstrated useful in the classical setting with innovative ones that suit the quantum setting.

We will start by exploring the applicability of the classical computing technique cause elimination, which is a debugging technique that formulates hypotheses (using inductive/deductive reasoning) by specifying a root cause for a bug under analysis. Then, data inputs and experiments are crafted to refute or prove the hypothesis. We argue that this approach suits quantum computing. Given the probabilistic nature of the QC programs[4], we plan to extend the techniques used for testing probabilistic programs running on CC [9] to the QC domain. Moreover, as this technique requires the program to be executed multiple times, possibly with different inputs, we will also explore strategies to generate inputs to quantum programs (by adapting the ever popular fuzzing techniques - which will require strategies to generate inputs for testing quantum programs). Finally, the multiple executions, leveraging the logs, we will propose semi-automated techniques to pinpoint the root cause of faulty programs[10].

All the techniques and approaches will not only be offered within an IDE but also seamlessly handle problems in both quantum and classical programs (i.e., hybrid programs) to ease adoption.

4. Conclusions

Computing in its current form is very much limited to address problems that scale linearly with the problem size. In turn, quantum computing is expected to be a true game changer, as it theoretically promises polynomial and exponential speedups. Numerous projects focus on hardware advances to enable this computation, but the awareness of this opportunity for the perspective of the user (e.g., developers of quantum programs) is still lagging behind.

In this position paper, we propose a roadmap to advance the state-of-the-art of design and development of quantum programs from the perspective of (quantum) software engineering. To achieve this vision, and similar to today's development of classical programs, we argue that the research community needs to focus its efforts in developing quantum algorithms and data structures as well as techniques of validate and verify quantum programs. These developments will offer off-the-shelve components and libraries that will help programmers to develop quantum programs without the need to fully understand the underlying (quantum

mechanics) technologies.

A key on the development of these approaches is to have the right abstraction such that fields ranging from business economics, computer science, computer engineering and electrical engineering are able to develop their applications. This abstraction will only be achieved if researchers from different domains collaborate amongst themselves.

Acknowledgements

That authors would like to thank Shaukat Ali, Marco Pistoia, and Koen Bertels for the countless discussions on defining a roadmap for Quantum Software Engineering.

References

[1] N. Wirth, Algorithms + data structures=programs, Prentice-Hall, Englewood Cliffs, N.J, 1976.

[2] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, J. L. O'Brien, A variational eigenvalue solver on a photonic quantum processor, Nature Communications 5 (2014). URL: http://dx.doi.org/10.1038/ncomms5213. doi:10.1038/ncomms5213.

[3] E. Farhi, J. Goldstone, S. Gutmann, A quantum approximate optimization algorithm applied to a bounded occurrence constraint problem, 2015. arXiv:1412.6062.

[4] A. Miranskyy, L. Zhang, On testing quantum programs, 2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER) (2019). URL: http://dx.doi.org/10.1109/ICSE-NIER.2019.00023. doi:10.1109/icse-nier.2019.00023.

[5] R. Rand, Formally Verified Quantum Programming, Ph.D. thesis, University of Pennsylvania, 2018.

[6] A. W. Cross, L. S. Bishop, J. A. Smolin, J. M. Gambetta, Open quantum assembly language, 2017. arXiv:1707.03429.

[7] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, B. Valiron, Quipper, ACM SIGPLAN Notices 48 (2013) 333–342. URL: http://dx.doi.org/10.1145/2499370.2462177. doi:10.1145/2499370.2462177.

[8] What are the Q# programming language and QDK? - Microsoft Quantum, https://docs.microsoft.com/en-us/quantum/language/?view=qsharp-preview, 2020. Accessed: 2020-10-03.

[9] S. Dutta, O. Legunsen, Z. Huang, S. Misailovic, Testing probabilistic programming systems, in: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2018, Association for Computing Machinery, New York, NY, USA, 2018, p. 574–586. URL: https://doi.org/10.1145/3236024.3236057. doi:10.1145/3236024.3236057.

[10] W. E. Wong, R. Gao, Y. Li, R. Abreu, F. Wotawa, A survey on software fault localization, IEEE Transactions on Software Engineering 42 (2016) 707–740.