The Dawn of Quantum Programming

Neil J. Ross
Department of Mathematics and Statistics, Dalhousie University
Halifax, NS, Canada

Seminal work in the field of quantum programming was done at the turn of the century but these investigations were often restricted to theoretical studies and toy programming languages [4, 5]. One reason for these limitations was the lack of enthusiasm from the broader quantum computing community. The experimental hurdles that separated us from quantum computers were so vast that the questions related to the programming of such hypothetical machines appeared irrelevant. The great experimental progress of the last few years, which recently culminated in small, but universal, quantum computers [3], has inverted this trend. While existing quantum devices remain limited in size, they hold a promise of scalability which warrants giving quantum programming a second thought.

Many questions which were hitherto considered either solved or peripheral have resurfaced with the recent developments in practical quantum computing. How should we *program* a quantum computer? To properly answer this question we need to identify the basic operations from which quantum algorithms are built in order to define programming languages that are natural and well-structured. Ideally, a quantum programming language will not only allow us to implement existing quantum algorithms but will also facilitate the discovery of new ones. Further, how should we *compile* a quantum program? Once a quantum algorithm is implemented in a programming language it must be translated into a sequence of operations that can be physically performed on hardware. This compilation must be executed in a way that does not offset the computational advantage we hoped to gain by using a quantum computer. We need our quantum compilers to preserve our quantum advantage.

The recent developments in practical quantum computing not only revive old questions about quantum programming they also raise new questions which quantum programming can help answer. Indeed, with medium-scale quantum computers on the horizon, it has become paramount to understand what to do with the first generation of super-classical quantum devices. In order to identify computational problems that can be solved with such devices we need to shift from an asymptotic understanding of quantum algorithms to a more concrete one. Scalable quantum programming languages and efficient quantum compilers can assist in understanding and minimizing the concrete cost of quantum algorithms.

Various quantum programming languages have been released in the last few years including LIQ$Ui\,|\rangle$ [8], Quipper [2], Scaffold [1], and, more recently, Q# [7]. All of these languages propose answers to the fundamental questions of quantum programming and were designed with the aim of addressing the challenges of practical quantum computing. In particular, all of these languages make it possible to express and reason about quantum algorithms of the size and type expected in real-world applications of quantum computing. In doing so, quantum programming environments can play an essential role in turning quantum computers from objects of science into instruments of scientific discovery.

ProjectQ [6], an open source software stack for quantum computing recently introduced by Damian S. Steiger, Thomas Haner, and Matthias Troyer, belongs to this new tradition of quantum programming environments inspired by the needs of practical quantum computing. ProjectQ provides a powerful circuit description language embedded in Python. The ProjectQ language has an intuitive syntax and many high-level operators acting on circuits. Therefore, a programmer can define circuits gate-by-gate but can also combine existing circuits into new ones. For example, a circuit can be controlled, iterated, or conjugated by another circuit. Further to the ability to describe complex families of circuits, ProjectQ provides a complete compilation framework. The ProjectQ compiler meshes decomposition and optimization passes which can result in very efficient implementations of quantum algorithms. Finally, the ProjectQ compilation framework is compatible with a variety of backends: circuit viewers, resource counters, simulators, and even IBM's quantum computer. ProjectQ is therefore a complete programming environment for quantum computing; one in which it is possible to express, optimize, test, and run quantum algorithms. As such, it represents an important contribution to the field of practical quantum computing.

References

[1] A. J. Abhari, A. Faruque, M. J. Dousti, L. Svec, O. Catu, A. Chakrabati, C.-F. Chiang, S. Vanderwilt, J. Black, F. Chong, M. Martonosi, M. Suchara, K. Brown, M. Pedram, and T. Brun. Scaffold: Quantum Programming Language. 2012.

[2] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron. Quipper: A Scalable Quantum Programming Language. *ACM SIGPLAN Notices*, 48(6):333–342, 2013.

[3] IBM Research. Quantum Experience. `http://www.research.ibm.com/quantum/`. Accessed: 2018-05-20.

[4] E. Knill. Conventions for Quantum Pseudocode. 1996.

[5] P. Selinger. Towards a Quantum Programming Language. *Mathematical Structures in Computer Science*, 14(4):527–586, Aug. 2004.

[6] D. S. Steiger, T. Häner, and M. Troyer. ProjectQ: an open source software framework for quantum computing. *Quantum*, 2:49, Jan. 2018.

[7] K. M. Svore, A. Geller, M. Troyer, J. Azariah, C. Granade, B. Heim, V. Kliuchnikov, M. Mykhailova, A. Paz, and M. Roetteler. Q#: Enabling Scalable Quantum Computing and Development with a High-Level Domain-Specific Language. 2018.

[8] D. Wecker and K. Svore. LIQ$Ui\,|\rangle$: A Software Design Architecture and Domain-Specific Language for Quantum Computing. 2014.