An open-source modular framework for quantum computing

An open-source modular framework for quantum computing

S. Carrazza[1,2,3], **S. Efthymiou**[3], **M. Lazzarin**[3], **A. Pasquale**[1,3]

[1] Dipartimento di Fisica, Università degli Studi di Milano and INFN Sezione di Milano.
[2] CERN, Theoretical Physics Department, CH-1211 Geneva 23, Switzerland.
[3] Quantum Research Center, Technology Innovation Institute, Abu Dhabi, UAE.

E-mail: stefano.carrazza@cern.ch

Abstract. In this proceedings we describe the current development status and recent technical achievements of Qibo, an open-source framework for quantum simulation. After a concise overview of the project goal, we introduce the modular layout for backend abstraction released in version 0.1.7. We discuss the advantages of each backend choice with particular emphasis on hardware accelerators for quantum state vector simulation. Finally, we summarize the primitives and models currently available.

1. Introduction

Quantum computing is a new paradigm whereby quantum phenomena are harnessed to perform computations. The current availability of noisy intermediate-scale quantum (NISQ) computers [1], combined with recent advances towards quantum computational supremacy [2, 3], has led to a growing interest in these devices to perform computational tasks faster than classical machines. Among many of the near-term applications [4, 5], the field of Quantum Machine Learning (QML) [6, 7] is held as one promising approach to make use of NISQ computers, including applications to evolving research fields such as High-Energy Physics [8, 9].

Nowadays, quantum processing units (QPUs) are based on two major approaches. The first one is based on quantum circuit and quantum logic gate-based model processors, as implemented most popularly by Google [10], IBM [11], Rigetti [12] or Intel [13]. The second employs annealing quantum processors such as D-Wave [14, 15] among others. The development of these devices and the achievement of quantum advantage [16] are indicators that a technological revolution in computing will occur in the coming years. However, in parallel to the development of QPU technology, we still have to perform classical simulation of quantum computing, which has been at the cornerstone of quantum research, to elaborate new algorithms and applications. From a theoretical perspective it serves as the basic tool for testing and developing quantum algorithms, while from an experimental point of view it provides a platform for benchmarks and error simulation.

Circuit-based quantum computers can be classically simulated using Schrödinger's or Feynman's approach [17, 18]. The former is based on keeping track of the full quantum state and applying gates via specialized matrix multiplication routines. The latter, inspired by Feynman's path integrals, can be used to calculate amplitudes of the final state by summing over different histories (paths). Schrödinger's approach is memory intensive as it requires storing the full

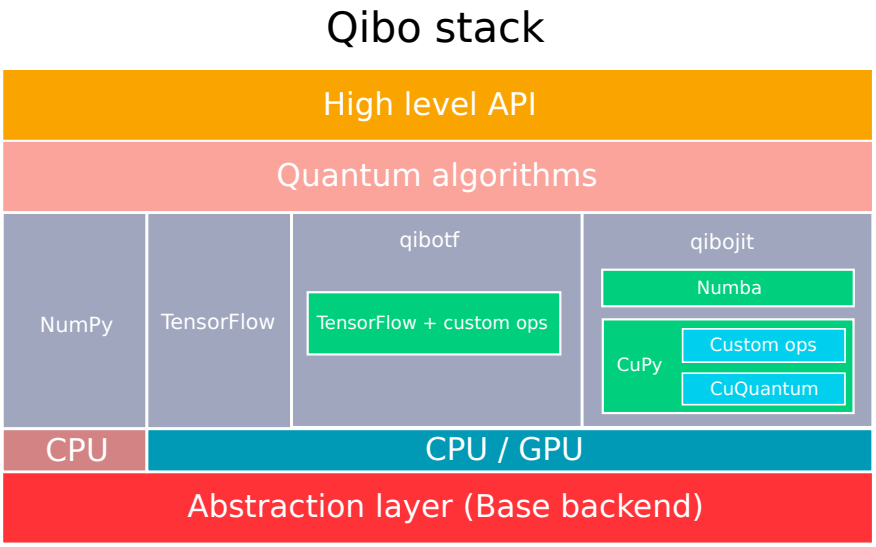

Figure 1. Schematic view of structure design in Qibo version 0.1.7.

quantum state consisting of 2^n complex numbers for n qubits, however its run time is linear on the number of gates in the circuit. Feynman's approach memory requirements scale linearly with the number of qubits and gates, however its run time grows exponentially with the number of gates [19, 20]. Hybrid methods exploiting both approaches to achieve a run time vs performance trade-off have also been explored [21].

Qibo [22, 23] is an open-source framework for quantum computing which supports general purpose quantum simulation based on Schrödinger's approach. The source code of Qibo, available at `https://github.com/qiboteam/qibo`, provides a high-level API for writing quantum circuits and gates, abstraction layers for simulation and hardware control backends and a collection of pre-coded quantum algorithms and research inspired examples. Its structure is visualized in Fig. 1 and will be discussed in the next sections using as reference the latest release version 0.1.7.

2. Quantum simulation backends

A system of n qubits is described by a state vector ψ of 2^n complex numbers, which represent the probability amplitudes in the computational basis. A gate targeting n_{tar} qubits can be represented as a $2^{n_{\text{tar}}} \times 2^{n_{\text{tar}}}$ complex matrix G. In Schrödinger's approach of quantum simulation, each gate is applied to the state via the following matrix multiplication

$$\psi'(\boldsymbol{\tau}, \boldsymbol{q}) = \sum_{\boldsymbol{\tau'}} G(\boldsymbol{\tau}, \boldsymbol{\tau'}) \psi(\boldsymbol{\tau'}, \boldsymbol{q}) \tag{1}$$

where $\boldsymbol{\tau}$ and $\boldsymbol{q}$ denote bitstrings of length n_{tar} and $n - n_{\text{tar}}$ respectively and the sum runs over all possibile bitstrings $\boldsymbol{\tau'}$ of length n_{tar}. From a computational point of view, such paradigm opens the possibility to use different techniques and hardware to achieve efficient simulation of the final state.

The Qibo package, distributed from PyPI[1] and conda-forge[2], is shipped with two basic simulators (`numpy` and `tensorflow`) which can efficiently simulate circuits of up to 20 qubits.

[1] `https://pypi.org/project/qibo/`
[2] `https://anaconda.org/conda-forge/qibo`

This base package provides an abstraction layer written in Python which defines an abstract backend class with a minimal set of methods for linear algebra manipulation and gate application. The `numpy` simulator is based on NumPy [24] primitives supporting only single thread CPU, while the `tensorflow` simulator replaces these primitives with those of TensorFlow [25]. These choices allow the simple execution of quantum circuits on multi-threading CPU and GPU configurations. The multiplications in Eq. 1 are implemented using the `numpy.einsum` and `tensorflow.einsum` methods that provide a generic algorithm with moderate performance. Despite the performance limitation, the `numpy` backend is relevant for cross-platform deployment. NumPy supports a high number of architectures, including `arm64`, allowing the `numpy` backend to be deployed in several contexts, such as laboratories developing QPUs, where servers do not always match the `x86_64` architecture. In addition, the `tensorflow` backend provides automatic gradient evaluation for gradient descent optimization. This combination allows the development of variational quantum circuits for quantum machine learning. This choice opens the possibility to develop novel hybrid classical-quantum models such as quantum generative adversarial networks [9].

Additional backends are available as add-on packages and provide higher performance for larger circuits. The `qibotf` package[3] was the first high performance backend included in the first Qibo release (0.1.0). It is based on TensorFlow custom operators written in C++ and CUDA. In contrast to TensorFlow primitives, custom operators perform in-place updates, *i.e.*, the state is not duplicated during circuit execution. This reduces both memory requirements and execution time. The main disadvantages associated with `qibotf` are the need to maintain C++ code, and the compilation of custom operators before execution which slows down development and complicates installation by reducing the target of potential devices that could benefit from pre-compiled binaries.

To address these issues, we developed the `qibojit` backend [26] package[4] which supports execution on multi-threading CPU, GPU, and multi-GPU configurations. The CPU part of `qibojit` uses custom operators like `qibotf`, which are now written in Python and compiled just-in-time using Numba [27]. The loop over the state elements is parallelized using OpenMP [28] via Numba's `numba.prange` method. Each element is updated according to the rule defined in Eq. 1. The proper indices for each update, which depend on target qubit index, are generated on-the-fly during the loop using fast binary operations. Gates that target multiple qubits and controlled gates can be applied similarly after modifying the index generation accordingly. The GPU part uses CUDA kernels that follow the same approach as CPU but are written in C++ and compiled just-in-time using CuPy [29]. The kernels are exposed to Python using CuPy's RawModule. Compatibility with CuPy also allowed us to incorporate to `qibojit` the recently released quantum simulation library cuQuantum [30] by NVIDIA. Exploiting Numba and Cupy capabilities simplifies the code without sacrificing performance. It also makes the installation on different platforms easier. Exhaustive performance benchmarks between the just-in-time and pre-compiled approaches for quantum simulation together with a technical explanation of techniques used to accellerate simulation performance are addressed in Sections II and III in Ref. [31].

3. Primitives and models

In Figure 2 we show the current components available when installing the base Qibo 0.1.7 package. At the current stage, the framework supports quantum simulation using circuits and annealing paradigms, which are most likely the two representations that can be deployed on real QPU devices.

The quantum circuit representation is composed of a set of primitives which allow the user

[3] `https://github.com/qiboteam/qibotf`
[4] `https://github.com/qiboteam/qibojit`

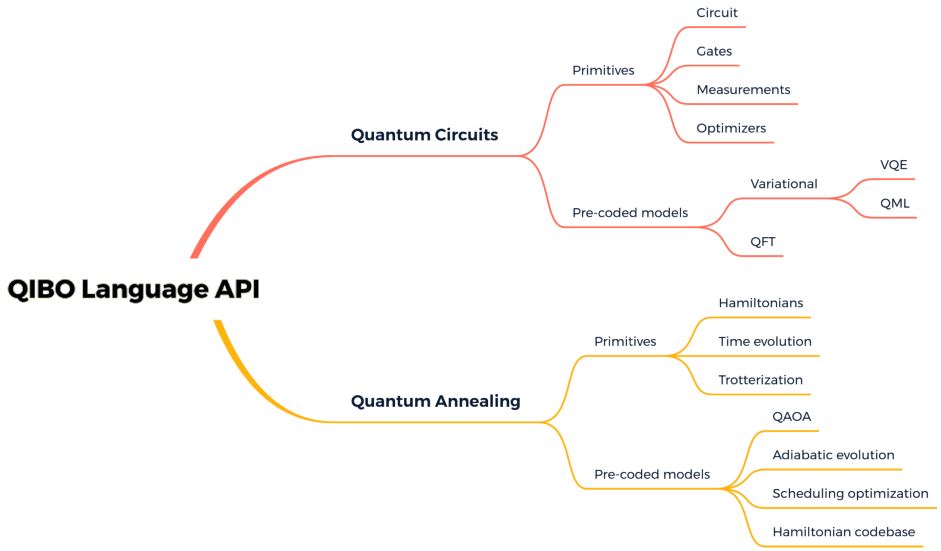

Figure 2. Models and primitives in Qibo version 0.1.7.

to allocate quantum circuits with gates based on single-, two-, and three-qubits. The simulation of measurements is provided through special sampling kernels for each backend. We have integrated classical optimization algorithms based on the approximate Hessian approach [32], evolutionary strategies [33] and gradient descent optimization through TensorFlow. In terms of built-in models, we provide variational gate layers which can be efficiently trained using the optimizers mentioned above. Furthermore, we include circuits such as the quantum Fourier transform (QFT), Grover's algorithm and the variational quantum eigensolver (VQE) for fast implementation and benchmark. Finally, we provide models for quantum machine learning, generative adversarial networks (style-qGAN) [9], quantum regressors [8] and several tutorials involving quantum classifiers [34].

Concerning the quantum annealing paradigm, we provide primitives for Hamiltonian representation, for both numerical and analytic representations. These objects are interfaced with a time evolution solver and the possibility to use the Trotter decomposition, as presented in Sec. 4.1 of [35]. We provide a large codebase of pre-coded Hamiltonians, such as non-interacting Pauli-X/Y/Z, transverse field Ising model, MaxCut, and Heisenberg XXZ. Furthermore, we include pre-coded models for adiabatic evolution [36], with the possibility to determine the best parametric scheduling function, and adiabatically assisted variational quantum eigensolver (AAVQE) [37], quantum approximate optimization algorithms (QAOA) [38], and the feedback-based algorithm for quantum optimization (FALQON) [39].

4. Outlook

The latest Qibo release (0.1.7) includes a modular approach to quantum simulation engines with support on multi-threading CPU, GPU, and multi-GPU hardware setups. We provide backends for multiple architectures, including those which are popular in experimental laboratories developing QPU technologies. The primitives and models implemented in the code are in continuous expansion and are driven by the requirements and feedback from users developing spin-off projects involving quantum computing.

In the future we are planning to explore the implementation of alternative approaches to state vector simulation (Schrödinger's approach). Furthermore, we will start testing the framework on real quantum hardware by expanding the current portfolio of backends to experimental setups.

References

[1] J. Preskill, Quantum computing in the NISQ era and beyond, Quantum 2 (2018) 79. doi:10.22331/q-2018-08-06-79.
URL http://dx.doi.org/10.22331/q-2018-08-06-79
[2] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, et al., Quantum supremacy using a programmable superconducting processor, Nature 574 (7779) (2019) 505–510. doi:10.1038/s41586-019-1666-5.
URL http://doi.org/10.1038/s41586-019-1666-5
[3] H.-S. Zhong, H. Wang, Y.-H. Deng, M.-C. Chen, L.-C. Peng, Y.-H. Luo, J. Qin, D. Wu, X. Ding, Y. Hu, et al., Quantum computational advantage using photons, Science 370 (6523) (2020) 1460–1463.
[4] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, et al., Variational quantum algorithms, Nature Reviews Physics 3 (2021) 625–644.
[5] K. Bharti, A. Cervera-Lierta, T. H. Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J. S. Kottmann, T. Menke, et al., Noisy intermediate-scale quantum (NISQ) algorithms, arXiv preprint arXiv:2101.08448 (2021).
[6] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, S. Lloyd, Quantum machine learning, Nature 549 (7671) (2017) 195–202.
[7] M. Schuld, F. Petruccione, Supervised learning with quantum computers, Vol. 17, Springer, 2018.
[8] A. Pérez-Salinas, J. Cruz-Martinez, A. A. Alhajri, S. Carrazza, Determining the proton content with a quantum computer, Physical Review D 103 (3) (Feb 2021). doi:10.1103/physrevd.103.034027.
URL http://dx.doi.org/10.1103/PhysRevD.103.034027
[9] C. Bravo-Prieto, J. Baglio, M. Cè, A. Francis, D. M. Grabowska, S. Carrazza, Style-based quantum generative adversarial networks for Monte Carlo events (2021). arXiv:2110.06933.
[10] Google Research, Google AI Quantum (2017).
URL https://research.google/teams/applied-science/quantum/
[11] IBM Research, IBM Quantum Experience (2016).
URL https://www.ibm.com/quantum-computing/
[12] Rigetti, Rigetti Computing (2017).
URL https://www.rigetti.com/
[13] Intel Corporation, Intel Quantum Computing (2017).
URL https://www.intel.com/content/www/us/en/research/quantum-computing.html
[14] D-Wave Systems, The Quantum Computing Company (2011).
URL https://www.dwavesys.com/
[15] D-Wave Systems, D-Wave Neal.
URL https://github.com/dwavesystems/dwave-neal
[16] F. A. et al., Quantum supremacy using a programmable superconducting processor, Nature 574 (2019) pp. 505–510. doi:10.1038/s41586-019-1666-5.
[17] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, H. Neven, Simulation of low-depth quantum circuits as complex undirected graphical models (2017). arXiv:1712.05384.
[18] J. Chen, *et al.*, Classical simulation of intermediate-size quantum circuits (2018). arXiv:1805.01450.
[19] E. Bernstein, U. Vazirani, Quantum complexity theory, SIAM Journal on Computing 26 (5) (1997) 1411–1473. arXiv:https://doi.org/10.1137/S0097539796300921, doi:10.1137/S0097539796300921.
URL https://doi.org/10.1137/S0097539796300921
[20] S. Aaronson, L. Chen, Complexity-theoretic foundations of quantum supremacy experiments, in: Proceedings of the 32nd Computational Complexity Conference, CCC '17, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, DEU, 2017.
[21] I. L. Markov, A. Fatima, S. V. Isakov, S. Boixo, Quantum supremacy is both closer and farther than it appears, arXiv preprint arXiv:1807.10749 (2018).
[22] S. Efthymiou, S. Ramos-Calderer, C. Bravo-Prieto, A. Pérez-Salinas, D. García-Martín, A. Garcia-Saez, J. I. Latorre, S. Carrazza, Qibo: a framework for quantum simulation with hardware acceleration, Quantum Science and Technology 7 (1) (2021) 015018. doi:10.1088/2058-9565/ac39f5.
URL https://doi.org/10.1088/2058-9565/ac39f5
[23] The Qibo team, qiboteam/qibo: Qibo. doi:10.5281/zenodo.3997194.
URL https://doi.org/10.5281/zenodo.3997194
[24] T. Oliphant, Guide to NumPy, 2006.
[25] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden,

M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL https://www.tensorflow.org/

[26] S. Efthymiou, S. Carrazza, qiboteam/qibojit: qibojit. doi:10.5281/zenodo.5071354.
URL https://doi.org/10.5281/zenodo.5071354

[27] S. K. Lam, A. Pitrou, S. Seibert, Numba: A llvm-based python jit compiler, in: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, 2015, pp. 1–6.

[28] The OpenMP development team, OpenMP website.
URL https://www.openmp.org/

[29] R. Okuta, Y. Unno, D. Nishino, S. Hido, C. Loomis, CuPy: A NumPy-compatible library for NVIDIA GPU calculations, in: Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS), 2017.
URL http://learningsys.org/nips17/assets/papers/paper_16.pdf

[30] NVIDIA, cuQuantum SDK (2021).
URL https://developer.nvidia.com/cuquantum-sdk

[31] S. Efthymiou, M. Lazzarin, A. Pasquale, S. Carrazza, Quantum simulation with just-in-time compilation (3 2022). arXiv:2203.08826.

[32] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 Contributors, SciPy 1.0: Fundamental algorithms for scientific computing in Python, Nature Methods 17 (2020) 261–272. doi:10.1038/s41592-019-0686-2.

[33] N. Hansen, yoshihikoueno, ARF1, K. Nozawa, M. Chan, Y. Akimoto, D. Brockhoff, Cma-es/pycma (Jun. 2021). doi:10.5281/zenodo.5002422.
URL https://doi.org/10.5281/zenodo.5002422

[34] A. Pérez-Salinas, A. Cervera-Lierta, E. Gil-Fuster, J. I. Latorre, Data re-uploading for a universal quantum classifier, Quantum 4 (2020) 226. doi:10.22331/q-2020-02-06-226.
URL http://dx.doi.org/10.22331/q-2020-02-06-226

[35] S. Paeckel, *et al.*, Time-evolution methods for matrix-product states, Annals of Physics 411 (2019) pp. 167998. doi:10.1016/j.aop.2019.167998.

[36] E. Farhi, J. Goldstone, S. Gutmann, M. Sipser, Quantum computation by adiabatic evolution (2000). arXiv:quant-ph/0001106.

[37] A. Garcia-Saez, J. I. Latorre, Addressing hard classical problems with adiabatically assisted variational quantum eigensolvers (2018). arXiv:1806.02287.

[38] E. Farhi, J. Goldstone, S. Gutmann, A quantum approximate optimization algorithm (2014). arXiv:1411.4028.

[39] A. B. Magann, K. M. Rudinger, M. D. Grace, M. Sarovar, Feedback-based quantum optimization (2021). arXiv:2103.08619.