Quantum Computing & HPC: Compilation Stack Similarities

Sonia Lopez Alarcon[1] and Anne C. Elster[1]

[1]Affiliation not available

April 16, 2023

Abstract

There is a lot of focus on how Quantum Computing as an accelerator differs from other traditional HPC resources, including accelerators like GPUs and FPGAs. In classical computing, how to design the interfaces that connect the different layers of the software stack, from the applications and its high-level programming language description through compilers, schedulers, down to the hardware, and gate-level, has been critical. Likewise, quantum computing's interfaces enable the access to quantum technology as a viable accelerator. From the ideation of the quantum application to the manipulation of the quantum chip, each interface has its challenges. In this column feature, we discuss the structure of this set of quantum interfaces, their many similarities to the traditional HPC compilation stack, and how these interfaces impact the potential of quantum computers as HPC accelerators.

Quantum computing will not replace classical HPC systems —at least not in the foreseeable future. However, there is currently a lot of research focusing on how they can be used as an accelerator for quantum simulations, Machine Learning applications (missing citation); (missing citation), optimization and combinatorial problems (missing citation); (missing citation), and other computationally expensive applications (missing citation). Quantum Computing grew from the birth of quantum information theory in 1970 and Benioff´s four publications in the early 1980´s that showed for the first time how quantum computers where theoretically possible [1] The first experimental quantum gates were implemented shortly after. IBM, Intel, Google, IonQ, Honneywell, Xanadu and many other large companies and start-ups are now all investing in advancing this technology, to the point that it is hard to keep up with the number of research papers being published.

The power of quantum computing stems from how densely they can represent information. This comes from the quantum superposition property — the linear combinations of two or more states, much like a combination of musical tones results in a new unique sound — and entanglement — the inexplicable correlations that happen between quantum bits (*qubits*). Interference is used to cancel portions of the superposition, similar to the use of noise canceling technologies in headphones. In addition, quantum gates are reversible, which means that the system preserves the information at any point of the execution. The theory says that, since information is not destroyed, application of the operands (aka, quantum gates) does not consume power. However, note that power is required to generate the operands and to keep a closed quantum state.

While traditional computing system store 0s and 1s, a two-qubit system has been claimed by IBM [2] to store the equivalent entangled state information of 512 classical bits, 10 qubits the equivalent of 16KB of classical bits, and the current large quantum computing systems with 100 and 280 qubits would need the number of bits equal to the number of atoms on planet earth and the universe, respectively.

[1] https://www.anl.gov/article/remembering-paul-benioff-renowned-scientist-and-quantum-computing-pioneer.
[2]https://www.ibm.com/thought-leadership/institute-business-value/report/quantum-decade

Quantum states are extremely delicate, and the challenge is to keep a system of qubits in its superposed and entangled state, and manipulate them in a controlled way. External interactions in all energy forms, even at the smallest scale, can easily make the state of the quantum system *fall out of coherence*, inducing noise as an error. This is a major challenge for implementing concrete quantum computers. Superconducting qubits need to be maintained at temperatures as cold as or colder than outer space in order not be susceptible to such errors. Recent technologies using photonic and diamond-defect designs try to overcome this. In the photonic case, the sensors still need to be super-cooled, whereas systems based on diamond-defects currently are limited to single-digit qubits [3].

The current technology used in quantum computing is known as the Noisy Intermediate-Scale Quantum (NISQ) systems, a term that was coined by John Preskill (missing citation). Small numbers of qubits with high error rates and limited connectivity define these systems. On these systems, only very specific applications that are hardly considered useful can outperform classical implementations. This is, however, a necessary step towards powerful quantum computing, with a high enough number of qubits to allow not only computational power, but also tolerance to error.

The progress is real, mainly and most crucially at the technology level, but also in all the other layers that separate the user from physical quantum system: algorithms, applications, programming models, and compilers (missing citation). Each of these layers is an *interface* that abstracts out the details of the layers below, and simplifies the development task.

The IEEE CiSE Leadership Department recently discussed the integration of quantum computing and HPC in a single software stack (missing citation). This time, we take a closer look at the software stack that bridges the gap between the quantum application, and the actual quantum systems leveraging quantum-mechanical properties.

[3]https://www.eetimes.eu/the-status-of-room-temperature-quantum-computers/ published March 20, 2023

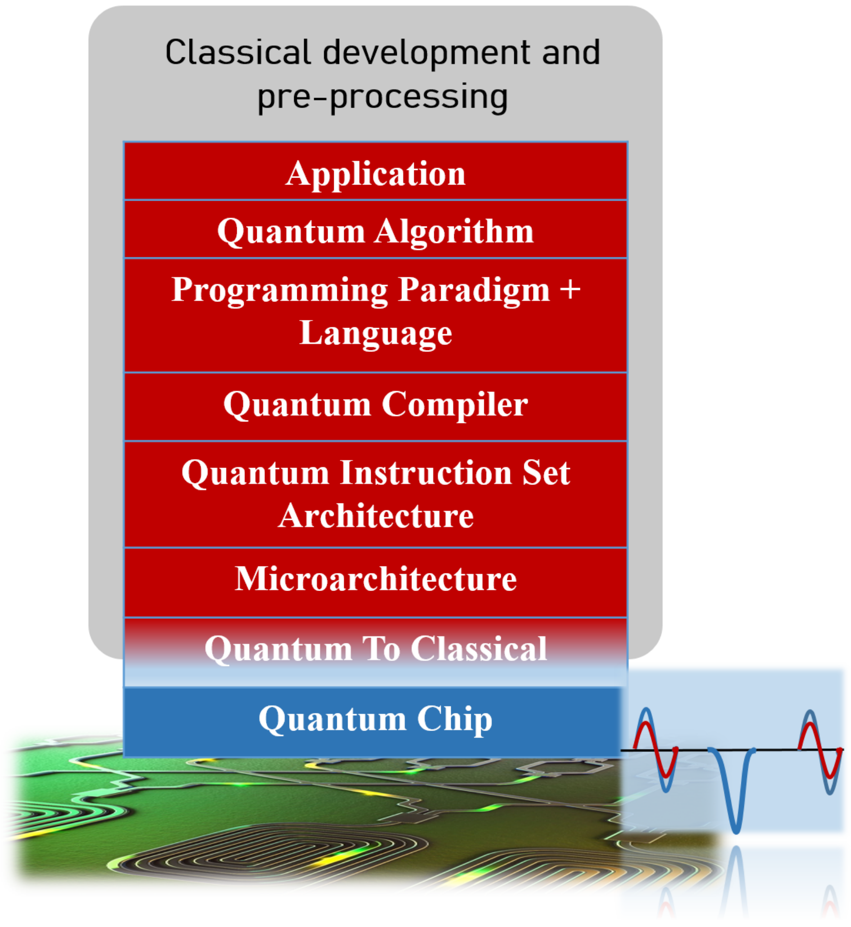
Classical development and pre-processing
Application
Quantum Algorithm
Programming Paradigm + Language
Quantum Compiler
Quantum Instruction Set Architecture
Microarchitecture
Quantum To Classical
Quantum Chip

Figure 1: The quantum computing software stack can also be envisioned as this stack of interfaces.

In classical computing, interfaces support the development steps, from the high level programming language description of an application to controlling electrons through semiconductor transistors. Similarly, quantum computing relies on a set of steps and interfaces (missing citation). The actual computation on the quantum hardware is only the final step, while the majority of the the development and pre-processing (D&PP) is done classically, with a quantum mindset, as shown in Figure 1.

The challenges of this D&PP are in no way negligible. The compiler in particular calls for a series of optimizations and graph problems that threatens the scalability of quantum computers.

QUANTUM COMPUTING INTERFACES

A quantum computer is a quantum system that evolves according to quantum-mechanical principles from an initial state to a final state. If all things went well, this final state contains the solution to a computational problem.

A stack of interfaces makes it possible to take a quantum application from ideation to reality. The software stack, and the interfaces needed to realize the quantum application is envisioned as a workflow in which the compiler plays a central role, as illustrated in Figure 2.

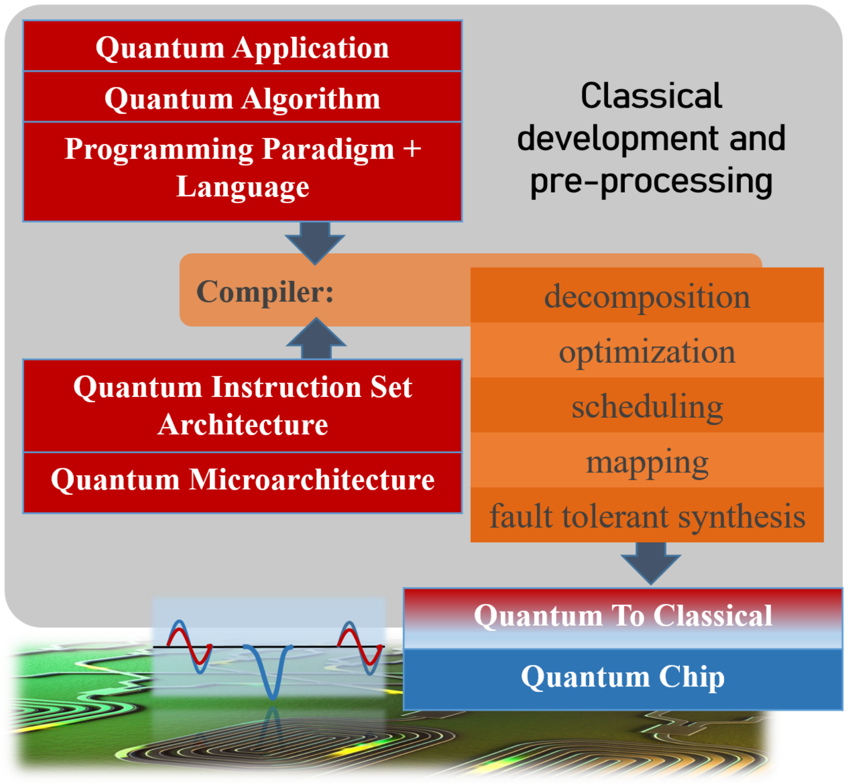

Figure 2: Quantum compilation takes in information from different interfaces. Through a number of steps, it generates the fault tolerant synthesis of the quantum applications.

A potential **quantum application** , e.g., identifying the maximum clique of a graph, makes use of a **quantum algorithm**, e.g., Grover's algorithm. The "only" quantum aspect of these two interfaces is the understanding of core and fundamental quantum properties, and how classical problems can be framed in a quantum context. This understanding is probably the greatest gap keeping the general scientific community from taking advantage of quantum acceleration at this point in time, even greater than the technology itself. It is still unclear which applications can be efficiently accelerated by quantum means.

Many scientists (mainly theoretical physicists) have described or envisioned these applications and algorithms as mathematical exercises on "pencil and paper." But to take these closer to an actual quantum implementation, a description on a suitable **programming language**, e.g., quantum specific Python extensions, should define the steps of the application and its quantum algorithms in a way that can then be **compiled** targeting a specific **quantum instruction set architecture** and **microarchitecture** .

The quantum instruction set architectures in their current form are far from being a set of general purpose instructions, but rather a sort of single- or two-qubit basic operands known as gates. Figure 3 depicts a basic circuit with several single- and two-qubit gates operating on a three qubit register. Each quantum computing technology has its own set of native gates. The implementation of these quantum gates is in the form of analog signals (microwaves, laser, or other depending on the technology) that act on the qubits, and that are generated and controlled by the microarchitecture. Also, the qubits' connectivity map is part of the system's architecture, since not all qubits can interact with all other qubits.

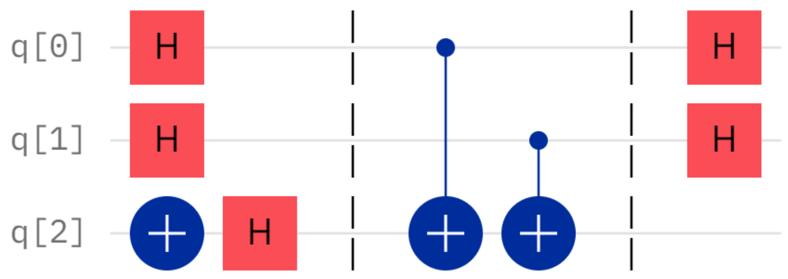

Figure 3: An example of a basic circuit, including a three qubit register, Hadamard, Controlled-NOT and NOT (+) gates build with IBM's Quantum Composer (missing citation). The computation proceeds from left to right, from initial to final quantum states of the three qubit register.

Therefore, with a description of the quantum application, typically as a collection of quantum gates known as a quantum circuit, and the quantum architecture and microarchitecture's information, the **quantum compiler** can generate the necessary information to control the quantum system: its initialization and quantum time evolution to the final quantum state on the **quantum chip** .

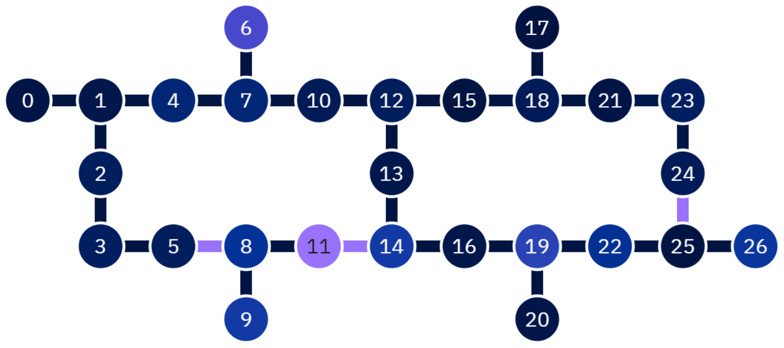

Figure 4: IBM Quantum 27-qubit Kolkata backend. The shades of color represent the quality of the qubits and the links among them (darker shade means better quality). This quality is based on different noise and error metrics. Different backends have different maps and noise levels (missing citation).

Notice that the D&PP down to the **classical-quantum** interface, does not involve any physical quantum interaction. Instead, all of this is done on classical computing systems.

QUANTUM COMPILATION

Compilation for a quatum computer system involves a number of steps that take the high level language description all the way down to generating the control signals. The **decomposition** of the high level gates breaks them down into the native gates of the architecture. For instance, Figure 5 represent a swap gate and its equivalent decomposition into three C-NOT gates. Sometimes, consecutive quantum gates cancel each other, or are commutative in the order of the execution. These **optimizations** are taken care of before **scheduling** the order in which the operands will take place, respecting all dependencies and exploiting parallelism when possible.

The **mapping** stage involves two operations: mapping and routing. In the quantum circuit high level description, gates act on quantum variables that we call logical qubits. These have to be mapped to physical qubits on the architecture's map. Then, as the execution evolves, the qubits that need to interact with each other through two-qubit gates are routed to physically adjacent qubits in the architecture's map.

For example, if Q1 and Q2 in Figure 5 were mapped originally to physical qubits 6 and 10 in Kolkata's map (Figure 4), Q1 would have to be re-routed to qubit 7, so it would be physically adjacent to 10. Qubit re-routing is done adding swap gates. Low connectivity is a critical problem in the current Noisy Intermediate-Scale Quantum (NISQ) systems (missing citation), with high noise levels, low coherence times and no error correction protocols enabled yet. In IBM's superconducting systems, the necessary swap gates (three CNOT gates each) for qubit routing do, on the other hand, accumulate link error, and increase the *depth* of the circuit in ways that often surpass the quantum coherence time of the system, resulting in too noisy of an output to be useful. Efficient routing algorithms do not only route to ensure correctness, but also to minimize noise.

Once the final circuit is built with all its optimizations and added swaps, and gates are scheduled, the compiler will generate the **fault tolerant synthesis** according to the system's microarchitecture.

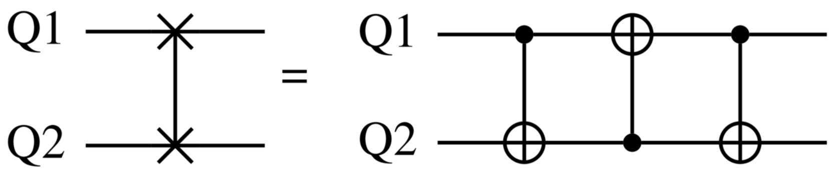

Figure 5: SWAP gate and its decomposition in three C-NOT gates.

Scheduling and mapping/routing are well known NP-hard problems that can be found in a plethora of other fields. The HPC community is well aware of the time complexity, memory and hardware resources required to solve these problems. Inputs are large graphs, such as dependency graphs or the connectivity map. The goal usually involves an optimization problem in which time, hardware usage, error or noise need to be minimized.

Quantum vs. Classical Compilation

Quantum and classical compilation processes have some analogies: starting from the dependency graphs, scheduling of operations and allocation of resources need to be performed. Those with experience in FPGA (Field Programmable Gate Array) acceleration may notice the resemblance with the High Level Synthesis full compilation stack. Figure 6 represents this stack, from the High Level Language description to the FPGA execution.

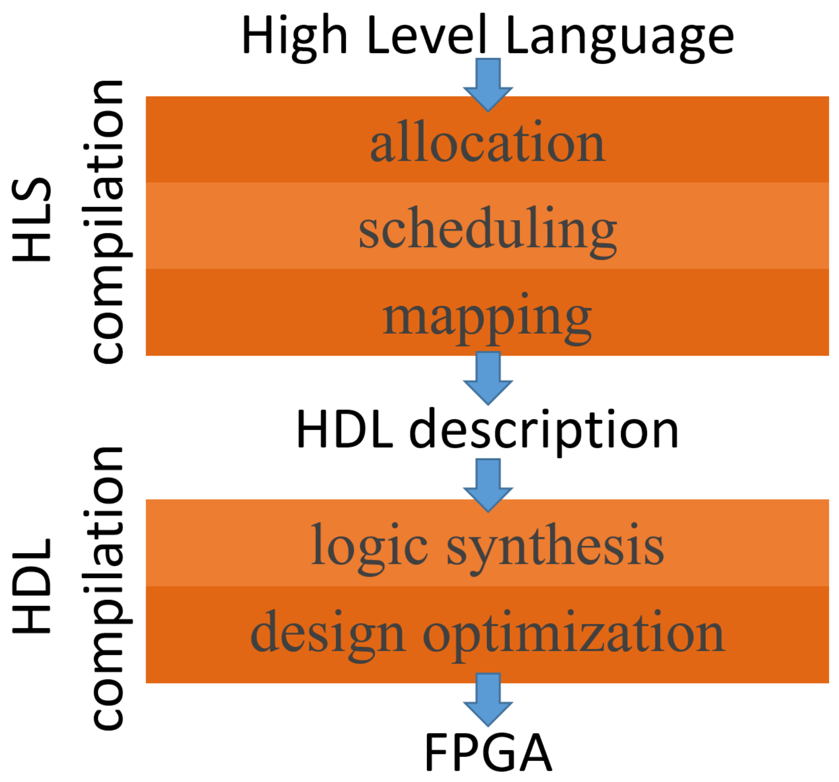

Figure 6: High Level Synthesis full compilation process: from a High Level Language description, an implementation in Hardware Description Language is generated (Verilog or VHDL) to then be synthesised and run on the FPGA

Similarly to the decomposition stage, allocation identifies the basic elements needed to implement the operations described in high level language. Then, operations are scheduled and mapped (aka. bound) to those elements. From the HDL description, the logic synthesis and necessary optimizations generate the files of 0s and 1s that will take care of the actual implementation on a field programmable gate array (FPGA).

FPGA users may have experience with the limitations of this process: compilation on-the-fly is prohibiting in terms of time, the HLL description of the computations cannot implement recursive calls, and may have issues with pointers and memory accesses, and in summary hardware design skills that are not accessible to all users are necessary. Whether using Hardware Description Languages (HDL) or High Level Synthesis, the implementation's description is hardware description, not software.

QUANTUM DESCRIPTION LANGUAGES

Setting aside the necessary understanding of quantum algorithms to be able to develop quantum applications, the quantum programming paradigm is the most immediate interface to the Quantum Computer user. The lexicon used around the description of quantum applications seems to indicate that we are describing hardware, just like VHDL or Verilog are used to describe FPGA hardware. First of all, because we call the description a "quantum circuit." Second, because the operands are referred to as "gates." Last, because the operations are described at a qubit-by-qubit granularity. There is no memory, no data types, and no general purpose, flexible instructions. None of this is compatible with software execution.

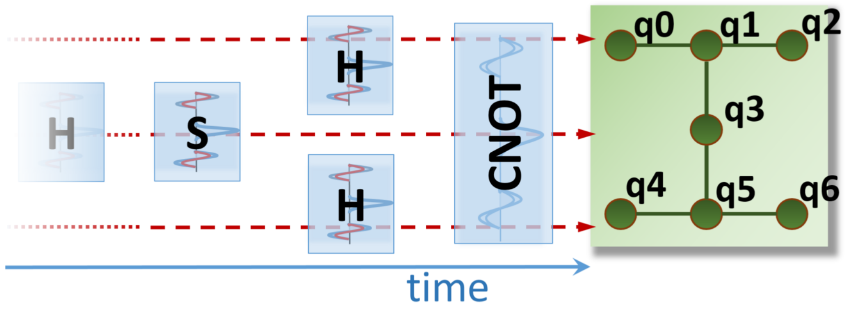

Figure 7: Quantum gates are sent to the quantum chip as control signals.

But if we take the term hardware as its literal translation of something that is hard-wired or physically palpable, this term does not apply either. Figure 7 gives an example of what really happens on execution of a quantum application, after all the steps in Figure 2 have been completed, and the quantum system finally kicks in. The quantum chip contains the "hard-wired" qubits. The quantum development and pre-processing has generated the sets of gates as analog control signals. Groups of these control signals arrive at the quantum chip to act on the qubits and alter the state of the quantum system. If, like in Grover's algorithm, the sets of gates need to be iterated through multiple times, the corresponding control signals can be just repeated as many times as needed. They are not real "hard-wired" gates.

The quantum programming models leverage high level languages, Python most commonly, to describe these extremely fine grained computations. The state of the quantum system contains the information. The quantum chip and its qubits act, actually, as a very short-lived memory upon which we need to operate before the system falls out of its delicate quantum equilibrium.

Although the quantum application is described using high level languages such as Python, much has to happen at the development level to give "quantum description languages" a software-like feeling. This includes the use of more generic APIs and libraries of computations. The implementation of reasonably long-lasting quantum memory is also a key piece that is missing in this picture, and that currently forces the start of every computation to long strings of operands just to initialize the states to the data that is going to be operated on.

CONCLUSIONS

Despite the many challenges, quantum computing holds real potential of accelerating certain applications, such as ML applications, combinatorial and optimization problems, and quantum molecular simulation. Proof of quantum acceleration for practical, real world cases will most likely have to wait until post-NISQ era is reached, with higher number of qubits, and the inclusion of error correction protocols. Most companies are looking at a five-year timeline (missing citation), primarily depending on technology advances.

Meanwhile, a solid stack of interfaces needs to be developed to support these future applications. In this article we discussed how the development and pre-processing of quantum applications entails a series of classical steps that can quickly become unmanageable, even more so given that the number of qubits and quantum gates required often can grow exponentially with the size of the problem — without even considering error correction mechanisms. The good news is that these are not new problems. Scheduling, mapping, routing, allocation and optimization problems are common in other fields, and heuristics can be applied to approximately solve these problems more efficiently. An efficient programming paradigm is yet to be defined, but the field can leverage the decades of experience of the HPC community at creating interfaces that bridge knowledge gaps, and that conceal the intricacies and challenges of the physical implementation.

ACKNOWLEDGMENT

The second author would also like to thank the Center for Geophysical Forecasting (SFI CGF) at NTNU and RCN (NFR proj. no. 309960) for their support during the preparation of this article.

Sonia Lopez Alarcon is an Associate Professor in Computer Engineering at Rochester Institute of Technology in NY, USA and also the Associate Editor of the IEEE CiSE Novel Architecture department. She received a PhD in Computer Engineering from Complutense University of Madrid (Spain). Her current research interest are on Quantum Computing and heterogeneous hardware solutions.

Anne C. Elster (IEEE Senior Member) is a Professor in Computer Science at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway and also the Associate Editor of the IEEE CiSE Novel Architecture department. She received a PhD in EE from Cornell University and was one of the original members of the MPI Forum. She has worked on GPU computing for computational science since 2006, and has a current research focus on autotuning and tools for enhancing applications related to geophysical forecasting. Contact her at elster@ntnu.no.

RELATED ARTICLES

- Contains a decent overview and several references: The Quantum Decade, Third Edition, IBV, IBM – https://www.ibm.com/thought-leadership/institute-business-value/report/quantum-decade
- The Status of Room-Temperature Quantum Computers, EE Times Europe, March 20, 2023, https://www.eetimes.eu/the-status-of-room-temperature-quantum-computers/
- Fuchs, F.G., Falch, V. & Johnsen, C. Quantum Poker—a game for quantum computers suitable for benchmarking error mitigation techniques on NISQ devices. Eur. Phys. J. Plus 135, 353 (2020). https://doi.org/10.1140/epjp/s13360-020-00360-5
- Quantum Computing Is the Future, and Schools Need to Catch Up: Top universities are finally bringing the excitement of the quantum future into the classroom, *Scientific American*, March 2023 [Online]. Available: https://www.scientificamerican.com/article/quantum-computing-is-the-future-and-schools-need-to-catch-up/ (URL)

References