Quantum Software Ecosystem Design

Achim Basermann, Michael Epping, Benedikt Fauseweh, Michael Felderer, Elisabeth Lobe, Melven Röhrig-Zöllner, Gary Schmiedinghoff, Peter K. Schuhmacher, Yoshinta Setyawati, and Alexander Weinert

Abstract The rapid advancements in quantum computing necessitate a scientific and rigorous approach to the construction of a corresponding software ecosystem, a topic underexplored and primed for systematic investigation. This chapter takes an important step in this direction. It presents scientific considerations essential for building a quantum software ecosystem that makes quantum computing available for scientific and industrial problem-solving. Central to this discourse is the concept of hardware–software co-design, which fosters a bidirectional feedback loop from the application layer at the top of the software stack down to the hardware. This approach begins with compilers and low-level software that are specifically designed to align with the unique specifications and constraints of the quantum processor, proceeds with algorithms developed with a clear understanding of underlying hardware and computational model features, and extends to applications that effectively leverage the capabilities to achieve a quantum advantage. We analyze the ecosystem from two critical perspectives: the conceptual view, focusing on theoretical foundations, and the technical infrastructure, addressing practical implementations around real quantum devices necessary for a functional ecosystem. This approach ensures that the focus is toward promising applications with optimized algorithm–circuit synergy, while ensuring a user-friendly design, an effective data management, and an overall orchestration. This chapter thus offers a guide to the essential concepts and practical strategies necessary for developing a scientifically grounded quantum software ecosystem.

Keywords Quantum computing · Software ecosystem · Hardware–software co-design · Software engineering

A. Basermann · M. Epping · B. Fauseweh · M. Felderer · E. Lobe (✉) · M. Röhrig-Zöllner · G. Schmiedinghoff · P. K. Schuhmacher · Y. Setyawati · A. Weinert
Institute of Software Technology, German Aerospace Center (DLR), Cologne, Germany
e-mail: elisabeth.lobe@dlr.de

I. Exman et al. (eds.), *Quantum Software*,
https://doi.org/10.1007/978-3-031-64136-7_7

1 Introduction

Over the past few decades, quantum computing has steadily garnered attention owing to its potentially transformative applications in various fields including cryptography [1], material science [2], linear algebra [3], and combinatorial optimization [4], among others. The possibility to vastly improve computational efficiencies in solving certain classes of problems, compared to classical computers, has driven significant interest and investment in quantum computing technologies from both the scientific community and industry.

In recent years the field has reached a new level of maturity, characterized by the development of more stable qubit systems and increased gate fidelities [5]. The emergence of quantum hardware platforms from academia and industry has underlined the significant strides made in this direction, creating a foundation for more advanced research and practical explorations in quantum computing [6]. However, it must be acknowledged that while substantial, these advancements are but the precursors to a fully fault-tolerant quantum computing potential.

Despite the progress, the current era of noisy intermediate-scale quantum (NISQ) devices [7] presents significant challenges, including limited qubit connectivity, low coherence times, and gate cross-talk. Moreover, the reliable physical fabrication of these devices, especially on an industrial scale, involves considerable hurdles: ensuring the purity of materials, achieving the precise alignment of nanostructures, and maintaining the ultra-low temperatures necessary for operation present ongoing challenges. Another problem is our limited understanding concerning the underlying principles of quantum algorithms, with a yet limited selection of algorithmic building blocks available, like the quantum Fourier transformation and the amplitude amplification. The development of a diverse and comprehensive portfolio of high-level algorithms is central to advancing the quantum computing field.

These factors naturally lead to the question: What is necessary to advance the field of quantum algorithms and how can we obtain meaningful results from these near-term quantum devices given the existing limitations? It is evident that, in the NISQ era, the fruitful utilization of quantum devices necessitates approaches that can effectively navigate the noise and errors inherent to current hardware.

In answer to this central question, we propose the necessity of creating an ecosystem that uses an interdisciplinary approach grounded in the principle of hardware–software co-design. This ecosystem requires the systematic development in software encompassing applications, algorithms, and compilers, and a robust technical infrastructure that is precisely aligned with the intricacies of existing and swiftly advancing quantum hardware. By establishing a framework where software development is intricately linked with hardware evolution, we aim to maximize the utility of quantum computing in its current NISQ stage and beyond. This approach does not exclude but rather complements hardware-agnostic abstractions that allow for more generic software development independently of the specific hardware.

In our view, a quantum software ecosystem comprehends all aspects in and around software designed for quantum computers, e.g., novel quantum algorithms designed for specific devices, optimized compilers, pre- and post-processing tools for results from quantum computations, and the technical integration into existing high-performance computing (HPC) environments. It includes the whole path from user perspective over access to actual hardware and, reversely, from the embedded hardware access to the general availability for different end users.

In this review we first describe a potential vision, how such a quantum software ecosystem interfaces with the potential end users and with the quantum hardware, in Sect. 2. We then analyze the requirements for an efficient ecosystem from the conceptual view, focusing on abstract requirements and methods, in Sect. 3. In Sect. 4 we are concerned with the technical implementation of such an ecosystem, and finally in Sect. 5 we give a concise conclusion and an outlook for the potential of such a scientifically constructed software ecosystem.

2 Quantum Computing Perspective

Future applications of quantum algorithms have the potential to provide novel efficient solutions in various sectors. This includes breakthroughs in material science, such as new superconductors or ultrafast memory, solutions for industrial size planning problems, applications in cryptography, or the design of new and more efficient drugs. In the following section we describe how a quantum software ecosystem supports these aims, by interfacing the applications with the quantum devices in a comprehensive and user-centered way.

2.1 Achieving the Vision Through the Quantum Software Ecosystem

As quantum computers continue to develop, it is plausible to predict a scenario where stakeholders, from academic researchers to industrial partners, gain access to quantum computational capabilities through cloud platforms. While such cloud access to quantum devices is already available for a limited number of platforms, the process is not yet streamlined and has various drawbacks due to the quantum device imperfections. However, such cloud-based access simplifies the challenges associated with operating and using quantum hardware, making it more feasible for a wider range of users.

At the heart of such a scenario, specialized quantum algorithms, devised by algorithmic developers, will be processed. In order to make these algorithms compatible with quantum hardware, specialized compilers, developed by experts in quantum software, will be crucial. These compilers will be responsible for translating high-level quantum logic into specific instructions, tailored for the distinct hardware platforms created by quantum hardware designers. Facilitating this process is the core responsibility of the quantum software ecosystem.

Furthermore, an integral component of this ecosystem will be the integration of quantum computers with classical systems. Fast embedded classical computers will process quantum-classical feedback algorithms within the coherence time of the quantum computer, especially those related to error correction. Additionally, HPC frameworks will be instrumental for algorithms that use parameterized quantum circuits, as these often require intensive computations to optimize parameters in tandem with quantum processors.

Another component shaping this ecosystem is the principle of hardware–software co-design. In this paradigm, not only is software adapted to optimally exploit the capabilities of the underlying quantum hardware, but the design of future quantum processors is also influenced by application-driven requirements. This bidirectional feedback ensures that hardware evolution remains attuned to the practical needs and challenges posed by real-world quantum applications. By closely intertwining the development processes of both hardware and software, the co-design approach seeks to accelerate the maturation and optimization of the quantum computing landscape.

After the computations are completed, users will receive their results via the same cloud interface. This closed-loop system aims to streamline the process of quantum computing, from input to result retrieval, while maximizing efficiency and user accessibility. The sustainability and success of this vision are inherently tied to the collaborative effort between quantum algorithm developers, compiler specialists, hardware builders, software engineers, and the users themselves.

2.2 Interested Parties and Their Requirements

Research and development in Quantum computing (QC) have accelerated dramatically in recent years. Due to its potential, efforts in QC have attracted different parties. They are classified as primary and secondary stakeholders. Primary stakeholders are stakeholders that directly contribute to the development of quantum computing as shown in Fig. 1.

1. End users: End users are individuals or organizations from different fields that use or adopt QC for various purposes, e.g., to speed up simulations for electric car batteries, to predict financial risk in insurance companies, or to optimize antenna patterns in radar technology. They are influenced by design and functionality features provided by the QC software researchers and the QC hardware developers. End users' expectations, values, and requirements must be considered to guarantee that the technology is effective and benefits them. The end users may not know how to write the algorithm and formulate the problem as a quantum program, but they can express it mathematically and are capable of post-processing the result of the computation as shown on the left panel of Fig. 1.

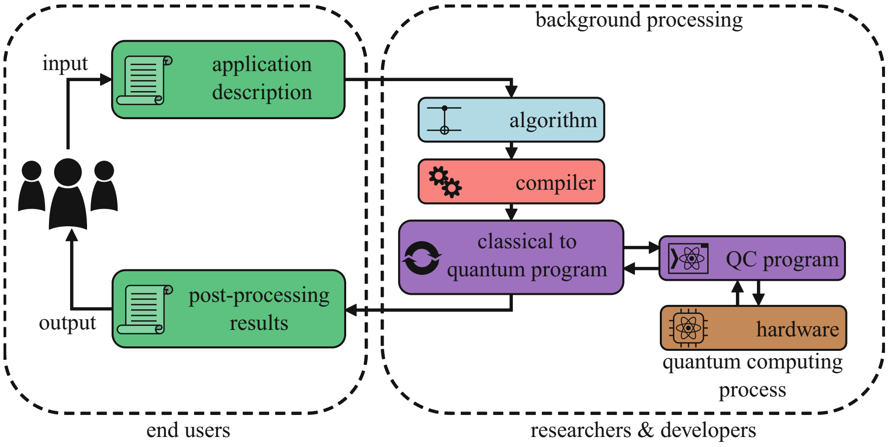

Fig. 1 Schematic diagram of the workflow and the stakeholders that directly use and develop quantum computing technologies

2. Researchers and developers: They are individuals and organizations that are directly involved in the development of and research on QC. Currently, research institutes and universities are the primary sources of this group, but also more and more large companies and start-ups participate in the development of QC. These vendors contribute significantly to the advancement of QC, for example, by developing hardware and software packages for industry and research institutions. Their role is shown on the right panel of Fig. 1 and includes algorithmic problem descriptions, compilation, software, and hardware development, such that the produced results can be post-processed and returned back to the end users. Hence, their work influences the design and development of technology; at the same time they must align with the goals of other stakeholders.

    (a) Software developers: These include private companies or research institutions that develop novel quantum algorithms, compilation schemes, and software interfaces between algorithmic solutions and hardware for QC. They also explore novel quantum computing architectures and investigate promising use cases for QC. Due to the noisy nature of current devices, the development has to take the low-level hardware properties into account to ensure optimal algorithm execution leading to unique design paradigms. In this context, it is important to have a clear and precise understanding of the performance of components and of the impact of physical quantum noise, which can be characterized by low-level benchmarks.
    (b) Hardware designers: The development of physical quantum computers is crucial. In many cases, hardware advancement is the bottleneck in the field of QC. Quantum computers are particularly sensitive to noise and errors caused by interactions with their surroundings. This can lead to an accumulation of errors, lowering computation quality. Thus, improving the fidelity of the

hardware operations is critical, even though noise can be tackled to some extent in software as well (see Sect. 3.7). Hardware manufacturers have a natural interest in making their devices available to a wide range of users. Some QC hardware is developed by private companies which might restrict information about the implementation details and restrict access to low-level control features, a fact that needs to be considered when developing software at the lower layers of the QC stack.

Secondary stakeholders are interested parties who can influence the future of QC but contribute indirectly to the workflow in Fig. 1.

1. Suppliers: They provide the necessary equipment and spare parts to build QC hardware. These stakeholders should consider requests from researchers and developers, whose involvement can shape the design and availability of technology. Semiconductor and chip manufacturers are two examples of this stakeholder group. The term "enabling technologies" is used in the context of QC to denote the development of products and enhanced manufacturing techniques that are not directly related to QC itself but will facilitate breakthroughs in QC and other fields. Therefore the suppliers play a crucial role in advancing the ecosystem.
2. Regulators and policymakers: They are responsible for the community's well-being and ensure that the developed technology boosts innovation. These governmental entities are also responsible for ensuring that QC aligns with society's values and needs, for example by motivating the development of QC to strengthen the economy and industrial advancement. Hence, they create laws and regulations for the development and use of QC. In many situations, they provide state funding for research and development and encourage enterprises to foster the growth of QC.
3. Investors: These are private funding sources that support research and development of QC. Investors are interested in the development of QC and expect a return on investment in the future. Investment in QC has increased significantly from US$93.5 million in 2015 to US$1.02 billion in 2021 globally [8]. Most investments are made for hardware, but there are also deals for software promising potential applications in the future.
4. Media: Media also play a significant role in the advancement of QC technology. They shape public opinion, hence raising awareness of QC development and its impact on society. They also convey the basic principles of this technology to the general public. Not only the potential, but also the growth of research, technology, startups, and investment is communicated through media.

Only the collaborative effort between all of these stakeholders will enable quantum computing to be established as a well-founded technology, where the quantum software ecosystem should support the communication and form the baseline for further advancements.

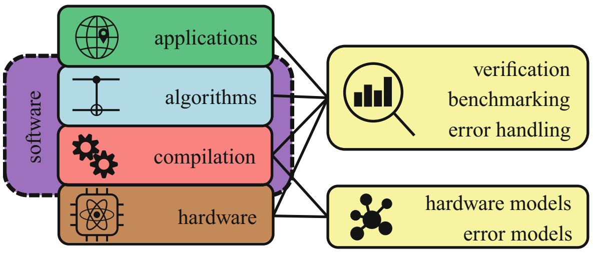

Fig. 2 Conceptual stack of the components necessary to solve problems using QC

3 Conceptual View

In this section, we examine the quantum software ecosystem from a more theoretical viewpoint, focusing on conceptually important ideas and abstracted QC concepts, which are the main area of scientific research on QC. This conceptual view includes various topics, as shown in Fig. 2.

At the top of this “stack,” i.e., on the side of the user, is the application or problem that needs to be solved, and on the bottom of the stack lies the hardware that executes the necessary QC steps. Those ends are connected by the software, including various algorithms and compilation schemes. In order to attain the correct results, it is necessary to handle the noise-induced errors emerging during the computation, which requires accurate error models for the hardware. One major challenge is the verification of the various parts of this stack. In the following, we look at each part of this stack and its role in the quantum software ecosystem.

3.1 Computational Paradigms

The development of a functional quantum computer is a central research goal these days. There exist different paradigms on how such a machine could look even on a conceptual level. In this section, we first review the basic principles of quantum mechanics on which all these quantum computing paradigms rely. Afterwards, we discuss the most prominent ones, namely the gate-based model and adiabatic quantum computation. Finally, we briefly mention a few alternatives.

3.1.1 Foundations of Quantum Computing

In this section, we outline the phenomenology that builds the foundation of QC without elucidating the rich mathematical framework of quantum mechanics that can be found in many textbooks [9, 10].

A quantum bit, or qubit for short, is a direct generalization of a classical bit with two additional, inherently quantum-mechanical properties: superposition and entanglement with other qubits. While a classical bit can only take one of the two states 0 and 1, a qubit can be in a superposition of both at the same time. Mathematically, the state of a single qubit can be expressed as

$$|\psi\rangle = a|0\rangle + b|1\rangle, \tag{1}$$

where $|0\rangle$ and $|1\rangle$ denote the computational basis states written in Dirac notation that is convenient in quantum mechanics and a and b are complex numbers with $|a|^2 + |b|^2 = 1$. The probability of measuring the state $|0\rangle$, i.e., a bit 0, is given by $|a|^2$ and analogously for $|1\rangle$ by $|b|^2$. After measurement, the state of the qubit collapses to only the parts in agreement with the measurement outcome, i.e., $|\psi_0\rangle = |0\rangle$ or $|\psi_1\rangle = |1\rangle$.

Since a and b are complex numbers, they each contain a phase ($a = |a|e^{\mathrm{i}\varphi_a}$). In quantum mechanics, only the phase difference $\varphi = \varphi_b - \varphi_a$ is relevant; hence the single-qubit state can be fully expressed by one probability and the relative phase, or equally by two angles. Thus, any single-qubit state can be visualized as a unit vector

$$|\psi\rangle = \begin{pmatrix} \sin(\theta)\cos(\varphi) \\ \sin(\theta)\sin(\varphi) \\ \cos(\theta) \end{pmatrix} \tag{2}$$

on the Bloch sphere, which is depicted in Fig. 3. This visualization is also useful to understand the concept of the computational basis: any two opposite points on the Bloch sphere can be chosen as the computational basis states $|0\rangle$ and $|1\rangle$ and changing the basis is equivalent to rotating the qubit state.

A superposition state needs to be initialized using classical information and after performing a measurement collapses to one of these two states, i.e., back to a classical bit. Therefore, the input and output are always restricted to classical bits, but during the computation the full space of superpositions can be exploited. It needs to be stressed that while a register of N classical bits can describe one of 2^N different states at a time, an N-qubit register can describe any state in a

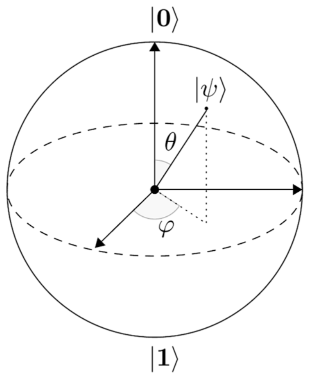

Fig. 3 Visualization of an arbitrary qubit state called the Bloch sphere. The computational basis states $|0\rangle$ and $|1\rangle$ are mapped to the north pole and the south pole respectively. A general state $|\psi\rangle$ is fully determined by the angles θ and φ. Any quantum gate on a single qubit corresponds to a rotation of the state on that sphere. Graphic taken from [11]

continuous region of a 2^N-dimensional vector space. As a consequence, qubits are tremendously more expressive than bits. Since each measurement can change the qubit state $|\psi\rangle$, consecutive measurements of the same qubit in different bases do not yield additional information, unless one prepares $|\psi\rangle$ anew for each measurement.

The second important property of qubits, quantum entanglement, is the ability of multiple qubits to interfere with one another such that their probabilities become correlated in a way that is not possible for classical bits. For instance, two qubits can be entangled in the state $|\psi\rangle = a|00\rangle + b|11\rangle$. When measuring the state of one of the qubits, the result automatically determines the state of the other qubit in the same computational basis, since, e.g., finding $|0\rangle$ for the first qubit collapses the full state to $|\psi_0\rangle = |00\rangle$.

It is noteworthy that any computation on the full qubit state $|\psi\rangle$ acts on all superposed states at the same time, e.g., on both $|00\rangle$ and $|11\rangle$. This is utilized by many powerful quantum algorithms that perform computations using precisely choreographed patterns of interference between superpositions of bit strings, which together with quantum entanglement realize the quantum computational efficiency. One needs to remember that measuring all qubits in a register collapses the carefully computed quantum state to a classical bit string, so care must be taken to prepare the final quantum state in a way that maximizes the probability of measuring the bit string that contains the relevant computational result.

Any natural or artificial quantum mechanical two-level system could in principle serve as a qubit, making the number of possible realizations incredible large. However, for fault tolerance a hardware platform needs at least to satisfy the DiVincenzo criteria [12]. It is necessary to have

1. A scalable physical system with well-characterized qubits
2. The ability to initialize the state of the qubits to a simple state
3. Long relevant coherence times
4. A universal set of gates
5. A qubit-specific measurement capability

These qualitative criteria point out immediately why building a functional quantum computer remains a challenge to date: on the one hand, satisfying criterion 3 requires decoupling the quantum system from any environmental disturbances. On the other hand, criteria 2, 4, and 5 demand direct physical access to the system and, therefore it is necessary to couple it at least to its measurement apparatus and some control electronics. This ambivalence makes quantum computers inherently error prone. As of now, no quantum system exists that fulfills all criteria equally, but recent quantum hardware has reached a level of maturity that allows for small-scale quantum computations. Platforms that have reached this level are dubbed NISQ devices.

3.1.2 Gate-Based Quantum Computing

In this section, we review the paradigm of gate-based quantum computing, which was the first quantum computing paradigm to be proposed [10]. Here, a quantum gate denotes the analog of a logical gate in classical computing. In the latter, there are only two possible gates on a single bit, namely the identity and the negation. By contrast, any operation corresponding to a rotation on the Bloch sphere, shown in Fig. 3, represents a valid quantum gate on a single qubit. Therefore, the set of valid quantum gates is uncountable even for that single qubit.

In order to realize an actually useful quantum computer, it does not suffice to consider single-qubit rotations. Instead, we need an N-qubit register, and we need to be able to apply multi-qubit gates on any set of qubits. Fortuitously, it turns out to be sufficient to have access to just a single maximally entangling two-qubit gate and to arbitrary single-qubit rotations to achieve universality [13]. In other words, any quantum gate applied to the N-qubit register can be realized as a sequence of these elementary gates. There are multiple universal gate sets. In many cases the QC hardware provides a basic set of gates, which ideally is universal.

One important consequence of quantum mechanical dynamics is that valid quantum gates must be unitary, i.e., the gate operations are represented by unitary matrices, which are reversible. Therefore, classical logic gates like the AND-gate, which has two input bits and one output bit, cannot be implemented directly on qubits without a second output qubit to ensure reversibility. Another consequence is that it is not possible to fully clone arbitrary qubit states, turning error correction by redundancy into a challenging prospect.

A sequence of quantum gates that solves a computational task composes a quantum algorithm. Quantum circuit diagrams have become established as a mode of representation, where the individual qubits usually correspond to horizontal lines on which gate operations are drawn (time runs from left to right) [10]. An example can be seen in Fig. 4.

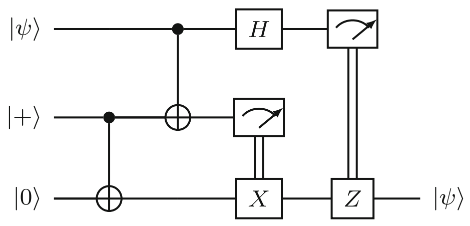

Fig. 4 An example of a circuit diagram, the most common way to represent quantum programs today. Horizontal lines correspond to qubits. Gates are represented by special symbols or boxes with labels. Double lines indicate classical information, which can represent results of the circuit. But they can also be used to condition the application of gates on measurement results, a technique called feed-forward

3.1.3 Adiabatic Quantum Computation and Quantum Annealing

Around 2000, a new computational concept based on quantum mechanical principles was developed, the adiabatic quantum computation (AQC) [14]. The underlying adiabatic theorem is a fundamental result in quantum mechanics, originally formulated in [15]. The AQC paradigm is different to the "conventional" quantum computing in the way that it does not provide a universal programmability straightforwardly in terms of implementing quantum gates to form quantum circuits. It rather represents a single algorithm whose input data can be varied. Nevertheless, the authors of [16] and [17] have shown that QC and AQC are equivalent in the sense that each can efficiently simulate the other. We briefly summarize the main background of AQC here, but for a more detailed review, we refer the reader to [18].

Given two related quantum systems, the rapid transfer from one to another might cause the system to change its state from their lowest-energy state, i.e., the ground state. However, by applying an adiabatic evolution process instead, which means a sufficiently slow transformation according to the adiabatic theorem, the system can remain in its instantaneous ground state with high probability. By encoding a mathematical optimization problem in the target quantum system, where the energy states represent the feasible solutions, we could thus obtain the minimal solution to the problem.

The first company that strived to build quantum systems based on AQC and made them commercially available was D-Wave Systems Inc. They implement the transverse field Ising model [19, 20], established by Ernst Ising, using superconducting loops to form qubits in a quantum system [21]. A current flow induces a magnetic flux in these loops, pointing either up or down or being in a superposition of both. Due to couplings of the loops by joints, the qubits interact with each other pairwise, where the strengths of the interactions can be adjusted with external magnetic fields. This way we can encode a quadratic function over binary variables, with linear and quadratic terms weighted according to the magnetic field strength. Finding the solution for such a quadratic unconstrained binary optimization (QUBO) problem is hard on classical computers. More precisely, its corresponding decision problem belongs to the class of NP-hard problems. This also means it relates to a large number of other problems, which can easily be transferred into a QUBO and therefore solved with these machines, at least in theory.

Although empirical studies like [22] provide hints that the output of the devices is in general close to the optimal solution, it is, however, not guaranteed to be achieved, nor is the success probability known in advance. Several physical restrictions prevent the realization of the theoretical concept of the adiabatic theorem, which only applies if ideal conditions prevail. One obstacle is, for instance, the shielding against environmental noise, which is never entirely achieved. Therefore, the term quantum annealing (QA) has been established, in reference to the classical heuristic simulated annealing, to distinguish the theoretical concept from the heuristic process performed by the corresponding devices [23]. In general, quantum annealing is repeated several times with the same configuration to obtain a sample set of solutions, and from those the best one is extracted.

3.1.4 Others

The gate-based model and quantum annealing are without question the leading quantum computing paradigms. However, there exist alternative paradigms that turn out to be computationally equivalent to these mainstream approaches. For example, a paradigm called one-way quantum computing is pursued in the context of photonic quantum computers [24]. As photons hardly interact in nature, they can have enormous coherence times (one detects coherent photons from other stars regularly), but it is a challenge to perform two-qubit gates between them for the same reason. In order to circumvent this issue, an elegant idea that relies on the Knill-Laflamme-Milburn proposal [25] is to prepare all entanglement non-deterministically first. If successful, then the computation is proceeded by measurements and single-qubit rotations only, i.e., by avoiding any further interaction [26]. However, functional one-way quantum computing has not been demonstrated yet.

Another universal approach for quantum computation is quantum random walks, or short quantum walks, a quantum mechanical analog to the classical random walk [27, 28, 29, 30, 31]. They can either be discrete-time [32] or continuous-time [33], and they are studied in the context of machine learning [34, 35] and photosynthesis [36]. Both versions can again be extended to non-unitary evolution by a joint generalization of quantum and classical random walks, called quantum stochastic walks [37, 38, 39]. In contrast to the completely coherent quantum walk, quantum stochastic walks give rise to a directed evolution.

3.2 Hardware

In 1936, Alan Turing proposed a conceptual blueprint for a universally programmable computer [40]. This event became the child birth of modern computer science. However, as the direct physical implementation of the "Turing machine" would be impractical, a huge variety of different hardware platforms were used to realize different computational models. This early time of modern computer science came to a sudden end with the invention of the transistor [41]. Since then, the development of classical computers has relied on the same key building blocks but miniaturizing them.

In close analogy to these early days of classical computing, there exists a huge variety of candidates for quantum computing hardware—the current status of quantum computer development resembles the construction of the Z3 by Konrad Zuse rather than building modern HPC systems. A rather broad overview of hardware platforms, including a classification with respect to the state of development,[1] can be found in [42]. In the following, we focus on the most developed platforms according to this study, which are depicted in Fig. 5.

[1] Due to the status of the year 2020.

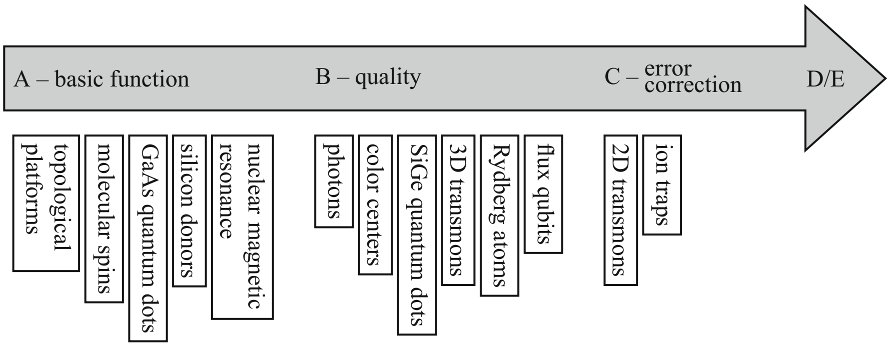

Fig. 5 State of development of different hardware platforms according to [42]. In this study, the platforms are classified into five different levels from satisfying the DiVincenco criteria (level A), to demonstration of high fidelities (level B), to the demonstration of quantum error correction (level C). The levels D (execution of fault-tolerant operations) and E (running fault-tolerant algorithms) have not been achieved by any platform so far

Generally speaking, there are two different classes of qubit candidates: natural quantum systems like neutral atoms, ions, or photons [25, 43, 44, 45], and artificial quantum systems like superconducting circuits or other solid state architectures [46, 47, 48, 49]. The state-of-the-art leading hardware platforms are based on trapped ions and planar transmons; the latter is a specific version of superconducting circuits. These platforms achieved the level of development C in Fig. 5, i.e., they allow for the demonstration of quantum error correction.

Superconducting integrated circuits are viewed as one of the most promising hardware candidates [50]. These circuits are put onto a chip that needs to be cooled to cryogenic temperatures, i.e., a few tens of mK, and they are controlled with electromagnetic fields in the microwave range. Even for this specific architecture, there is a variety of different qubit designs. However, all these designs share the same key ingredient, namely the Josephson junction [51]. This is a nonlinear element leading to a non-equidistant energy spectrum of the circuit. This property is crucial to address two quantum states as the computational states individually.

There are two mainstream types of superconducting qubits, i.e., charge qubit-[52, 53, 54] and flux qubit [55, 56, 57]-derived designs. To date, the primary representative of charge-derived qubits is the planar transmon, due to its suppressed sensitivity against charge noise at the cost of small anharmonicities in the level splittings [54, 58]. It operates at a sweet spot with rather long coherence times and a good reproducability of the qubits. The main benefit of planar transmons is their rather straightforward scaling in qubit numbers; the challenge here is to maintain the controllability of the individual qubits and to keep high-fidelity operations when scaling up. Transmons are typically considered for implementing gate-based quantum computing. One draft of a corresponding chip is shown in Fig. 6, where the planar transmons are arranged in a two-dimensional square lattice with nearest-

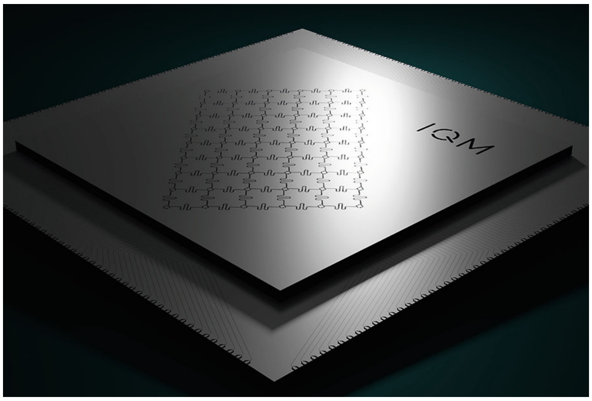

Fig. 6 Sketch of the KQCircuits chip design by the company IQM Quantum Computers (courtesy of IQM Quantum Computers)

neighbor interactions. Control and readout lines are connected to the qubits from below.

Flux qubits consist of superconducting loops that are interrupted by an (effectively) odd number of Josephson junctions. Their computational states are encoded in the magnetic fluxes that are induced by clockwise and anticlockwise circulating currents. By design, they share a lot of similarities to superconducting quantum interference devices (SQUIDs) [59]. Flux qubits can be coupled easily via mutual induction with coupling constants up to ultra-strong coupling if needed. This makes them an auspicious candidate for quantum annealing, and possibly for specific quantum simulation applications. In comparison to planar transmons, flux qubits are easier to couple, but it is harder to reproduce them reliably.

Apart from technical challenges, one of the main drawbacks of superconducting qubits is their limited connectivity: only nearest neighbors are directly coupled and hence two-qubit gates can only be applied between them directly. If a gate-based quantum algorithm needs gates between qubits that are not physically connected, one needs to perform the desired logical gate by swapping the qubit state through the intermediate qubits. This process produces a serious overhead in circuit depth. For quantum annealing, the limited connectivity becomes even more serious, because general optimization problems require strongly connected problem Hamiltonians. Therefore, embedding the desired problem Hamiltonian on the actual hardware becomes a nontrivial task [60]. Moreover, as superconducting qubits are artificially made, every single qubit has slightly different parameters than the others, an issue that needs to be tackled by optimal control theory [61].

With up to about 20 qubits, the best performing quantum computer is a chain of isotopically pure ions in a linear Paul trap.[2] The ions are trapped in an ultrahigh vacuum using electromagnetic fields in a quadrupole geometry such that they form a one-dimensional crystal [62, 63]. No cryogenics are needed; the trap operates at room temperature. In contrast to superconducting qubits, the ions in the trap

[2] Named after Nobel laureate Wolfgang Paul.

are coupled via the long-range Coulomb interaction, leading to a natural all-to-all connectivity of the qubits. In comparison to other hardware platforms, the relevant coherence times are high, and the gate quality is excellent.

Unfortunately, the design of the linear Paul trap does not allow for a scaling to large qubit numbers for two reasons. On the one hand, adding more and more ions into the trap deforms their arrangement; the ions start to form two-dimensional structures instead of a well-controlled chain. On the other hand, an effect called frequency crowding becomes more and more dominant, such that the system becomes uncontrollable [64]. Therefore, the main challenge for trap ion-based quantum computing is the scaling to larger qubit numbers. One ansatz is to combine several linear Paul traps via photonic links [65]. Here, the quantum information needs to be converted from the ions in the trap to photons that are transmitted through a fiber, and then it is converted back to the ions in another trap. This process makes quantum computing with trapped ions enormously slow, because every single conversion only succeeds with limited probability. A different strategy is to use two-dimensional surface traps instead of linear Paul traps [66]. Here, the second dimension is used to shuttle the ions during the computation to different zones on the chip, depending on their current purpose (performing a gate, readout, etc). In the gate zone, the surface trap mimics the linear Paul trap with its advantages locally. A photograph of such a surface trap is shown in Fig. 7. However, surface traps have not yet been able to demonstrate the same quality as linear Paul traps.

In this section, we discussed the benefits and drawbacks of the furthest developed hardware platforms to date, namely superconducting circuits and ion traps. However, as the field develops rapidly, other platforms may take over in the future. But even in this case, the substantial challenges to build functional quantum computers will probably remain during the coming decades [7]. Therefore, any near-term quantum software ecosystem needs to incorporate the specific hardware restrictions that are present or that are expected to remain in the near future. For example, one requires additional compilation techniques to run a desired quantum

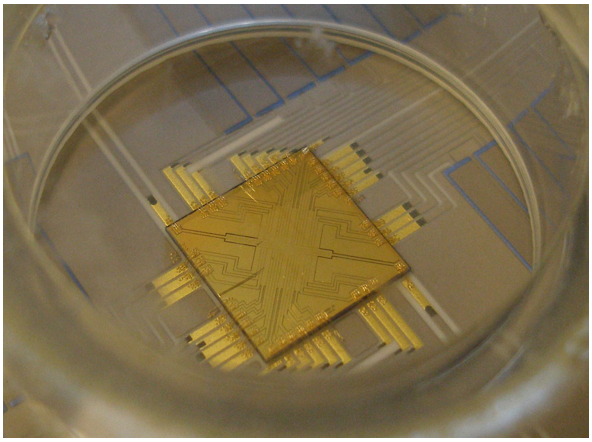
Fig. 7 Photograph of the surface trap chip design by the company eleQtron (courtesy of eleQtron)

algorithm on a superconducting qubit platform due to its limited connectivity, as on ion-trap platforms with natural all-to-all connectivity. Conversely, if a given quantum algorithm can be easily embedded on the connectivity graph of the superconducting chip, then this platform might be preferential because of the larger qubit numbers that can be achieved. In the long term, as soon as universal fault-tolerant quantum computers are realized, the necessity to keep track of the specific hardware limitations by designing a quantum software ecosystem will become less and less important.

3.3 Applications

Quantum computers have enabled advancements in a range of applications, starting with well-established domains such as database search and factorization using Grover's and Shor's algorithms. These have a proven potential in enhancing search capabilities and disrupting traditional cryptographic methods, respectively, but require a level of fault tolerance not yet reached on quantum devices.

Beyond these utilities, quantum machine learning is emerging as a noteworthy area of application [67, 68], enabling advancements in categorization, learning tasks, and the solution for partial differential equations. However, it is on the intermediate timeline where quantum simulation and optimization are drawing heightened attention. Quantum simulation facilitates the study of quantum systems, promising more accurate modeling of atomic and chemical processes, with applications in material science, quantum chemistry, and drug design. In parallel, quantum optimization provides avenues for solving complex problems more efficiently, finding its relevance in logistics, finance, and more.

In the forthcoming sections, we narrow our focus on quantum simulation and optimization, as these represent the realms where quantum computing is expected to offer significant advantages in the near term.

3.3.1 Simulation

Digital quantum simulation (DQS) represents a notable application for future quantum computers, focusing on simulating quantum systems with universal quantum computers. Richard P. Feynman originally suggested this application [69], later formalized by Lloyd [2]. DQS is of particular significance for studying quantum materials like superconductors and topological insulators, which prove challenging for classical simulations.

Emergence, described as the rise of new system properties from the fundamental interactions of its components, has been evident in quantum phases and is directly connected to the existence of strong quantum fluctuations and entanglement. Traditionally, the examination of such phenomena relied on resource-intensive experiments, which explored only a limited range of parameters, including material composition and external electromagnetic fields. Theoretical modeling and simulation can significantly conserve resources and is pivotal for advancing material science. Yet, simulations of quantum models on conventional computers face challenges due to the exponential scaling with system size. Classical simulations on modern HPC hardware are capable of describing non-equilibrium dynamics in quantum dots [70], of 1D quantum systems [71], as well as 2D systems [72, 73], but with strong limitations in the simulatable system size.

DQS employs quantum computers to efficiently simulate quantum systems. However, the current state of DQS struggles to match the capabilities of conventional HPC. Advancements in the present NISQ hardware require innovative quantum algorithms like the variational quantum eigensolvers (VQEs) [74], which capitalize on the increased expressiveness of quantum computers [75]. Ongoing research is centered on assessing the strengths and weaknesses of various hardware platforms concerning their potential DQS applications [76].

3.3.2 Optimization

Optimization problems appear in all fields where resources are limited, for instance in engineering, economics, computer science, and many others. The development of efficient solution methods and answering the question as to whether these actually exist is the essential part of the research in mathematical optimization and complexity theory. A very important and well-studied class of problems are the NP-hard ones, which are, loosely speaking, those problems that cannot be solved efficiently using classical computation. This situation cannot be alleviated simply by increasing the computational resources of classical computers. This naturally calls for the exploration of different, more powerful computational models. And the hope is that quantum computation steps into the breach due to properties of superposition, entanglement, and quantum parallelism.

As explained in Sect. 3.1.3, quantum annealing is a tailored method to solve discrete optimization problems. Several studies have shown the practical feasibility of this approach in different research areas, e.g., for the optimization of flight routes [77], flight gate assignments [78], and satellite scheduling [79]. However, due to their heuristic nature, the actual practical advantage of the quantum annealers over dedicated classical approaches, including approximation algorithms and heuristics, is still under discussion.

Besides the optimization-tailored QA, also algorithms for the gate-based quantum computing concept have been developed, like quantum approximate optimization algorithm (QAOA) or Grover search, which we elaborate in the next section. However, due to currently too limited available resources, their performance on interesting industrial applications still needs to be evaluated in the future [80]. To investigate the capabilities of all such approaches systematically, they need to be integrated into a full software environment that allows for quickly formulating different applications and for benchmarking the results of the quantum devices against several classical approaches.

3.4 Algorithms

The application cases described in Sect. 3.3 can also, in principle, be solved on classical computers. In order to gain a speedup over these classical approaches by using quantum computers, efficient quantum algorithms are necessary. While many promising algorithms already exist, there is active work on expanding the existing toolbox. A quantum software ecosystem must provide a library of algorithms that end users can access and must also support the development of new algorithms for domain and quantum experts.

The development of novel quantum algorithms faces two main challenges: Currently, there is much less experience in realizing quantum algorithms as software than for classical algorithms, and in order for quantum algorithms to be viable, they need to provide a significant asymptotic speedup over existing classical algorithms. The following section provides a selection of important quantum algorithms, many of which provide super-polynomial speedup. A more extensive overview can be found in [81].

3.4.1 Powerful Algorithms for Fault-Tolerant Devices

Table 1 lists some of the most promising quantum algorithms [82], which are expected to provide a quantum advantage on fully fault-tolerant QC. One such algorithm is Shor's algorithm for the prime factorization of large integers with super-polynomial speedup compared to the classical counterpart. Shor's algorithm is based on the quantum Fourier transformation and connected to the more general class of hidden subgroup problems, which include e.g., discrete logarithms and Gauss sums. Grover's algorithm searches through an unsorted list with a polynomial speedup. The quantum phase estimation algorithm approximates eigenvalues of a given Hamiltonian. Furthermore, a quantum computer can efficiently perform quantum time evolutions and SLE can be solved with the algorithm by Harrow et al. [3]. A variety of other quantum algorithms, such as the Deutsch-Jozsa algorithm [83], the

Table 1 Examples of promising quantum algorithms for fault-tolerant QC

Algorithm	Application case	Complexity	Classical complexity
Shor	Prime factorization of integer with N bits	$\mathcal{O}(N^2 \log N)$	$\mathcal{O}(\exp(1.9N^{1/3} \times (\log N)^{2/3}))$
Quantum Fourier Transform	Fourier transform with N amplitudes	$\mathcal{O}((\log N)^2)$	$\mathcal{O}(N \log N)$
Grover	Unsorted search on N items	$\mathcal{O}(\sqrt{N})$	$\mathcal{O}(N)$
Quantum Phase Estimation	Eigenvalues of unitaries up to error ϵ	$\mathcal{O}(1/\epsilon)$	$\mathcal{O}(N^2)$
Harrow-Hassidim-Lloyd	Solving SLE with N eq. and condition number κ	$\mathcal{O}(\kappa^2 \log N)$	$\mathcal{O}(\kappa N)$

Table 2 Examples of promising quantum algorithms for NISQ devices

Algorithm	Application case	Complexity	Classical
VQE	Eigenenergies and -states	Heuristic, often $\mathcal{O}(N^p)$	$\mathcal{O}(e^N)$
QITE	Ground state preparation	For highly local Hamiltonians $\mathcal{O}(N^p)$	$\mathcal{O}(e^N)$
QAOA	Combinatorial optimization	Heuristic, potentially $\mathcal{O}(N^p)$	$\mathcal{O}(e^N)$

Bernstein-Vazirani algorithm [84], and the Simon algorithm [85], have been found as well, but won't be discussed here in detail.

3.4.2 Hybrid Algorithms for Noisy Intermediate Scale Devices

Fully fault-tolerant quantum computers are not expected to be built in the near future. Therefore, great effort is put into researching efficient algorithms for NISQ devices, where the focus lies more on achieving quantum advantage over classical devices than on the best asymptotic performance. Many of these algorithms are heuristic and an asymptotic speedup is expected in special cases [86]. Some promising approaches in this area are listed in Table 2.

One general strategy to bring useful quantum algorithms on NISQ devices is hybrid computation, where only the part of the problem that gains most from quantum hardware is solved on such, while the remaining problem is solved on a classical device. One example of this is variational quantum algorithms (VQAs), most famously VQEs [87]. The idea of VQAs is to use a parameterized circuit on the quantum processor to prepare highly entangled states in the exponentially large Hilbert space and perform measurements on them. The classical processor evaluates the measurement results and adapts the parameters of the quantum circuit in order to improve the result. For instance, VQEs minimize the energy to find the ground state. Various adaptions of this approach are being researched at the moment, such as searching for excited states by optimizing the energy to be in a certain range or by enforcing orthogonality to the ground state. Furthermore, ground states can be prepared efficiently for highly local Hamiltonians by using quantum imaginary time evolution (QITE) [88].

The QAOA [89] is used to solve combinatorial problems by encoding them as a Hamiltonian with bit strings as representations of the possible solutions. The QAOA applies time evolution of a mixer Hamiltonian and problem Hamiltonian in alternation to find the bit string that minimizes the problem Hamiltonian expectation value.

A central challenge with performing these optimization algorithms in polynomial time is the risk of converging to local minima. It is important to extend the scope of these algorithms and facilitate an infrastructure where a hybrid compiler (see Sect. 3.6) can efficiently select which parts of a given problem to solve on the quantum device with quantum speedup.

3.5 Software Engineering

The goal of software engineering is the efficient development of high-quality software through scientific methods and precise processes. In this context, we understand software to be a structured collection of program code, documentation, quality assurance measures, artifacts, and, where applicable, other data required to execute the programs. All software is written to perform specific tasks that can be described in the form of user stories: A user wants to achieve a goal with the software. The value of the software therefore lies in the efficient and reliable achievement of these goals.

Quantum software fits the above scheme just as well [90]. At this level of abstraction, the only difference is that quantum software contains parts that are executed on a quantum computer. As described above, quantum computers are particularly suitable for difficult problems, and the applications institutions, such as the German Aerospace Center (DLR),[3] are particularly interested in having a strong interdisciplinary character and will have a large scope. An efficient, structured approach and an integrated quality assurance strategy will therefore be essential in the near future.

In the following, we take a closer look at the aspects of software engineering where we recognize specific requirements of quantum computing or which, in our view, are particularly important in this context.

3.5.1 Requirements

A particular challenge in the development of software for quantum computers is the collection and specification of requirements [91]. It can differ significantly from classical software requirement engineering [92].

The first step is to describe the primary requirements. In our experience, this is done in collaboration with domain experts who often have little experience with quantum computers. Finding a common understanding of the problem to be solved is tedious, but always worthwhile. Subsequently, a precise mathematical formulation must be worked out that allows the mapping of the application to an existing quantum algorithm or the development of a new one.

The joint elaboration not only helps the software engineer to find a solution approach, but also gives insights into quantum computing to a wider circle of interested people. This experience building within the organization, but also within the ecosystem as a whole, is something we have recognized as having its own value [93, 94].

Secondary requirements arise from the primary ones, e.g., requirements on the size of the system via the input data. The requirements must be considered together

[3] www.dlr.de

with the expected limitations of the hardware, a step that admittedly often leads to disillusionment and requires several iterations at this point. For instance, at DLR there is a huge gap between the problem sizes that quantum computers can handle and the massive computing tasks that arise in engineering questions. However, we must and can already set the course for future advantages in our fields of application. Despite or precisely because of the current hardware-related limitations, scalability must always be considered in quantum software development. There is great value today in demonstrating an algorithm that can solve small instances of difficult problems if it "only" needs to be scaled in the future; see [95, 96, 97] for the example of Shor's algorithm [1]. In contrast, it seems questionable to implement a highly optimized algorithm that does not even theoretically scale to large instances.

3.5.2 Software Design

Software design is always about defining the architecture, components, and their interfaces. In the design of quantum software, a dimension is added that is very important. It is necessary to decide which parts of the program are to be calculated on a conventional computer and which on a quantum computer. In this context, one also speaks of a quantum processing unit (QPU), which can take over specific tasks. Not every task is well suited for a QPU, and it does not currently look as if quantum computers will completely replace conventional processors.

Once it has been determined what is to be computed where (which includes in particular the choice of a quantum algorithm as discussed above), a specification of the data exchanged between the conventional and quantum parts must be made. A hardware-aware concept is required in order to feed data of a certain accuracy from a classical computer system reliably and accurately into a specific quantum circuit. Speed requirements here depend on the integration of the quantum hardware with the classical hardware and on the algorithm to be executed. Some hybrid algorithms require communication between the classical and the quantum systems within the coherence time. The challenge here is to define abstract layers in the software design so that software solutions for reliable and accurate data communication between classical and quantum system are at least partially reusable.

A conceptual separation of software into hardware-specific and -agnostic parts increases the reusability of the software we develop. It is important to understand that, although we use very low-level methods to get the most out of our quantum computers, we aim to develop software and methods that are useful in the long term. Therefore, reusability is an important criterion.

Interfaces must be defined for the transfer of data. At present, there is practically no distinction between program code and data on the quantum computer. Input data is transferred via program code for preparing the data [98], which can be very hardware-specific; see [99, 100] for examples on ion traps. We expect that future abstractions will facilitate data transmission.

The development of suitable data types on quantum computers is still in its infancy. A lot of research and standardization is still needed here. However, it is already apparent that, even for integers, the type of encoding has a major influence on the performance of quantum computers [101]. Possible choices are amplitude encoding or basis encodings like binary encoding, one hot encoding, and domain wall encoding [102]. More ways to encode classical data into quantum states are considered in the context of machine learning; see, e.g., [103]. They affect the performance mainly due to the strong noise of current models, so any form of resource optimization can help a lot.

In many engineering applications, decimal fractions are of course required, which, depending on the required resolution, generate a very high resource requirement by today's standards (measured in number of qubits). It can therefore be worthwhile to choose an algorithm that is formulated in data types that fit well with a quantum computer.

Finally, a good design process for quantum software includes simulations of the program and, if possible, test runs on available hardware. It allows challenges to be identified and the design to be adapted if necessary. A rigid approach here is even more doomed to failure than in conventional software design.

3.5.3 Models and Representation

Let's take a look at current ways of representing quantum software, or rather program code for quantum computers. At the moment, mainly low-level descriptions are used. Even in most recent publications we are still on a level where quantum algorithms are described via elementary gates and quantum circuits. Internally, these circuits can be represented as a list of gates, directed acyclic graphs (DAGs), path integrals/phase polynomials, or decision diagrams. Low-level languages such as OpenQASM [104], cirq [105], and qiskit [106] have become established as descriptions by a user and as interfaces between tools. Despite some attempts to create more high-level quantum programming languages, e.g., Q# [107], Silq [108], or qrisp [109], none of these is currently widely used (for various reasons). In the long term, however, there is no way around the introduction of more powerful language constructs in our view. It will be crucial that these find a natural way to represent the special capabilities of quantum computers. Although perhaps only years of programming experience will make natural programming languages for quantum computers possible, we want to support developments in this direction at an early stage.

In the context of compilers in particular, intermediate representations (IR) are also introduced as an intermediate level between the abstraction layers of the programming language and the machine language. Examples are QIR [110] and QSSA [111]. The formulation of quantum-specific optimization steps on this level is a subject of current research, which we will discuss in Sect. 3.6. We should also mention that other established tools of conventional software design are currently translated to and tried in the context of quantum computing, e.g., the unified modeling language (UML) [112].

3.5.4 Software Testing

Software testing is part of the software development process that aims to ensure the quality and reliability of the software. There are different types of tests, and common categories are unit tests (testing small components), integration tests, functional tests, and acceptance tests (checking fulfilment of requirements). Tests are artifacts (code or instructions) that are executed automatically or manually. In contrast, verification relies on formal proofs which employ static code analysis, and benchmarking is concerned with the quantification of the performance of software and hardware. We look at verification and benchmarking in more detail in Sect. 3.8. We emphasize that testing is about finding programming bugs, not hardware errors, whose treatment we discuss in Sect. 3.7. However, we investigate how the methods developed for handling hardware errors can also be adapted to testing.

Given the above definition of testing it is clear that future software for powerful quantum computers will also need to be tested. It is important to do basic preliminary work already now, before the hardware allows complex software to run. And research in this direction has indeed started [113, 114, 115, 90]. This ensures that the reliability of software does not become a bottleneck in future developments of QC. It is particularly important because in QC the transition from low-level circuits to high-level programs mostly still lies ahead of us. And testing is an exciting research topic in the field of quantum software engineering because quantum-specific phenomena have to be taken into account.

It is obvious that facts like the no-cloning theorem [116] are obstacles in testing programs. Classical approaches that often use copying implicitly need to be adapted in order to apply them to quantum software. The fact that in general measurements in quantum theory disturb the observed system also complicates state monitoring. This severely affects the possibilities for runtime tests on quantum computers (see e.g., [117, 118]), and further research in this direction will be necessary.

Furthermore, what constitutes a typical error is quite different between classical and quantum programming. Due to the difference in the computational model there are even programming errors that are not meaningful in classical programming, e.g., when they affect only the phase of the state. It is therefore necessary to conduct studies on what bugs are typical in quantum programs [119, 120]. Such studies can be very programming language-specific, i.e., tailored toward Q# [121]. Only with knowledge about typical bugs is it possible to then develop good tests that detect as many of them as possible. A useful tool here is the creation of benchmark collections, as well as the automatic generation of test cases.

It is necessary to define tests that circumvent the abovementioned quantum-specific challenges, and are still meaningful. And this leads to further research questions, such as the definition of meaningful measures for the significance of tests. Once more and more quantum software is written, guidelines for writing reliable code and informative tests which are based on the above research will be very useful.

3.6 *Compiling*

3.6.1 Gate-Based Quantum Computing

Like conventional computers, quantum computers also implement a finite set of elementary basic operations, the gates already mentioned above. Different sets of gates have become accepted for the description of quantum circuits [122, 123, 10]. If a gate set enables an efficient approximation of any unitary operation, we call it a universal gate set. By efficient here we mean that the new length of the circuit scales polynomially with the original circuit length when switching from any other gate set.

On the one hand, convenient gate sets are used for the theoretical description of quantum algorithms. These sets in general contain significantly more gates than necessary, are useful in the context of fault tolerance, and might also contain larger, undecomposed blocks. On the other hand, each hardware platform implements different, sometimes very limited, gate sets. A major restriction results, for example, from limited connectivity, which means that the two-qubit operations provided are not possible for every pair of qubits. However, some hardware platforms, in particular ion traps, provide native multi-qubit gates that allow this issue to be circumvented; see also Sect. 3.2.

The transition from one description of the quantum circuit to another is called *transpiling*. Specifically, the transition from a general unitary to a set of elementary gates is called *synthesis*. Both transitions are core tasks of a compiler. Furthermore, the compiler, just like its conventional analog, has the task of customizing the output to the specific hardware as best as possible. In summary, the requirement of a compiler is producing correct, efficient, and hardware-compatible output, as explained in more detail below.

The typical compiler architecture can be divided into individual steps (passes), which are connected in series as a pipeline where each step transforms the quantum circuit. The best order is not obvious and passes can also be repeated at a later point in the compilation. Typical transformation steps include:

- **Synthesis**. Larger operations need to be decomposed into a universal set of basic gates. Small operations can be decomposed optimally with regard to a certain cost function, while synthesis of larger operations will not yield optimal solutions in general.
- **Routing**. The circuit needs to be rewritten in a way that contains only gates that are natively supported by the hardware. In particular, multi-qubit gates can only act on qubits that can interact physically. Even qubits that might not be part of a calculation can mediate the interaction.
- **Optimization**. The overall circuit can be optimized with regard to some cost function as well. Here the input and the output are both decomposed circuits. We discuss this point in more detail in the following.

Various objective functions for circuit optimization are conceivable and are used. For example, the number of certain gates (e.g., controlled-Not gate, T gate), the depth of the circuit (related but not identical to the runtime), or the expected noise on the final state can be minimized. Of course, one can also try to maximize algorithm-specific performance, e.g., the probability of success. The problem of optimizing a circuit with respect to a particular objective is generally very difficult [124, 125, 126], such that there is no efficient algorithm to find the global minimum except for small circuits. Some approaches are based on meet-in-the-middle [127] or satisfiability (SAT) solvers [128]. For larger circuits only heuristic algorithms are feasible; see, e.g., [129]. For the optimization passes there are promising research approaches to transfer the conventionally established methods based on IR to quantum compilers.

Deciding whether the result of the compilation is indeed efficient on actual devices is not obvious. Since the global optimum is generally not known, one can only compare the result with other reference compilers. However, this comparison depends strongly on the circuits. It is important to use balanced benchmark suites, for example, the Arline Benchmark suite [130]. Further developments in this direction are foreseeable.

The development of compilers of hybrid programs has a major impact on the possibilities for optimization. Such hybrid compilers are compilers that do not generate pure quantum circuits, but executable code on conventional computers that contains calls to a QPU [131]. This results in strong optimization potential because the compiler can automatically decide whether the calculations are better computed on the QPU or on a conventional computer. Furthermore, it is even possible to apply hybrid simplification rules, which, e.g., move individual operations from the quantum circuit to conventional pre- or post-processing [132], where they can be combined and simplified with established methods. The goal is to leave only the essence of the quantum algorithm in the quantum circuit. These hybrid simplification rules in particular can benefit from the established concept of IR.

Another motivation for hybrid compilers is a closer coupling of the central processing unit (CPU) and the QPU. In particular, hybrid quantum algorithms such as VQAs [87] benefit greatly from an efficient coupling of conventional and quantum systems. Here, experience in GPU programming (e.g., CUDA [133]) can be built upon. In the future, abstract language constructs should simplify and unify the use of different hardware architectures. QPU and CPU codes are developed in a common project folder, where calls to the QPU are controlled via synchronous and asynchronous commands. Efficient interfaces and protocols must be developed for uploading data and code to the QPU and downloading measurement results. The QPU code may not only contain quantum operations but also increasingly complex dynamic operations that directly process measurement results and influence subsequent quantum operations. This feed-forward approach opens up exciting possibilities for new experiments, and it is essential in the measurement-based model of quantum computation [134]. The close coupling of a CPU and a QPU is flanked by development work aimed at integrating quantum computers into HPC environments; see [135, 136]. The experience with these prototypes will influence the necessary standards.

As mentioned, we require the correctness of the compiler, i.e., a proof that the output corresponds in functionality to the original input, possibly in human-readable form. However, it is known that the general equivalence test problem for quantum circuits is not efficiently solvable, as it is in the class of QMA-complete problems [137], which are, loosely speaking, those problems that are hard to solve for quantum computers. This means that we have little hope of proving the correctness of the final result. What we can do instead is to prove the correctness of the process. The compiler is correct if it only applies correct transformations. And for each individual transformation, it is possible to show correctness. When we speak of heuristics in the compiler pipeline, we mean procedures that do not necessarily lead to improved circuits but which nevertheless output a correct circuit in every case.

In addition to the described methodology, some approaches attempt to prove the equivalence of circuits. Although they suffer from an exponential increase in resources (time or memory), they can still deliver results for "simple" circuits. We refer the interested reader here to the literature [138, 139, 140].

3.6.2 Quantum Annealing

Although quantum annealing (QA) is a different computational model and therefore poses its own challenges, compiling in a certain sense is also needed here: quantum annealers can only process a very specific optimization problem, in case of D-Wave, a restricted version of the Ising problem [60]. Programming such devices essentially means providing the problem-defining parameters. However, exemplary applications from industry and research, cf. Sect. 3.3.2, show that there is in general no trivial way of obtaining these parameters. Several transformation steps from the original problem formulation to the native one of the device are required. A compiler handling these different abstraction layers would make the technology available for various users with different levels of expertise in QA.

From a mathematical point of view, the step from an arbitrary discrete optimization problem to a general Ising problem is solved and can be done using a set of standard methods. However, a complete software suite implementing this is not yet available. Nevertheless, toolboxes like the D-Wave Ocean SDK [141] or `quark` [142] already support users with utility methods. But further expansion of the software suites and conceptual advances are necessary. For instance, the recent research on the reduction of combinatorial optimization problems has mainly focused on any kind of (polynomial) reduction and not on the optimal one in a certain sense, e.g., in the number of resulting variables, which would be advantageous regarding the limited resources of current quantum computational devices. Furthermore, the actually implemented Ising problem is not a general one but faces further restrictions, such as a specific non-complete hardware graph and a limited parameter precision. This causes the reformulation of the original problem to be a nontrivial task and demands that a “compiler” implement all the necessary steps and hide the complexity from the application-focused users.

The two main transformation steps are the graph embedding and the parameter setting. Unfortunately, the first step, the embedding of the original problem graph into the hardware graph, has appeared to be a computationally hard problem, in particular, as hard as the problem D-Wave’s annealers are capable of solving [143]. Therefore, in practice, heuristic methods need to be applied to circumvent this bottleneck [144, 145]. In the second step, the hardware-native Ising problem has to be formulated based on the found embedding. If this step is not done correctly, we will not be able to analyze the actual performance of the quantum device itself, because the success probability might be suppressed due to a wrongly formulated problem. Recently, a new formulation has been developed that provides an embedded Ising problem which provably corresponds to the original problem and meanwhile optimizes its parameters with respect to the machine precision [146]. Based on this recent and future theoretical work, the compilation software has to be steadily improved and extended, and the full software ecosystem has to be able to adapt to these changes.

3.7 Error Handling

It is essential that errors caused by imperfect hardware are considered in the software stack, because the amplitudes and phases of the qubits are not discrete. Additional steps or layers are necessary to protect the information against this unavoidable noise introduced by the hardware. This section briefly sketches the main concepts in this field of research and provides references to more in-depth introductions.

We distinguish three categories of error-handling strategies. First, techniques that start directly at the hardware level and attempt to reduce the noise level [147]. This includes, for example, dynamical decoupling [148], where special control pulse sequences are used to eliminate the disturbing influence of the environment. Second, techniques that encode the quantum information into subspaces that do not couple to the environment and are therefore not affected by decoherence introduced by the environment [149]. Third, taking the noise of the quantum computer into account for the compilation can lead to circuits that are less prone to noise. For example, we investigated which decompositions of a common multi-qubit gate introduce the least amount of noise [150].

Furthermore, some post-processing steps on the classical measurement data are aimed at removing the noise [151, 152]. For example, zero-noise extrapolation [153] and readout error mitigation [154] have proven effective in some applications. The term error mitigation has come to refer to these types of techniques. We emphasize that they do not avoid errors, but try to eliminate the errors afterwards. Finally, methods on the error correction codes are aimed to suppress the noise to any degree [155, 156, 157]. We explain error mitigation and error correction in more detail in the following sections.

3.7.1 Error Models

Simple models for describing noise on quantum computers may only depend on a single parameter or a few parameters. They can be found in any textbook on quantum information, e.g., [10]. The depolarizing noise model is often used and can be interpreted in such a way that with a certain probability (the parameter of the model) the state of the system is replaced by white noise. Of course, this is a poor representation of the real experiment. However, we have found that it is often very suitable for a first qualitative picture of the effect of noise. Other simple models are the bit-flip, phase-flip, and amplitude damping channels.

A more precise description of the noise is possible with a Pauli channel [10], where every tensor product of Pauli operators can appear as an error. These errors can occur with different probabilities. A complete description of the error channel via Kraus operators [10] is also possible. The free parameters of this model can be determined via process tomography in the experiment [158, 159]. It is a complex procedure that does not scale well with the system size but obtains complete information. It is worthwhile, for example, if one wants to obtain a very precise picture of a single gate of a quantum computer. Instead of determining the parameters experimentally, one can also use "realistic" noise models. In this case, one tries to understand and model the physics of the process as well as possible. Often the parameters of the model have a physical interpretation. This approach is very hardware-specific and requires an exact fit of the model to the experiment. However, it also offers the chance to draw conclusions about necessary hardware improvements from model calculations, which is very helpful in the paradigm of hardware–software co-design.

We follow yet another approach in which the noisy process is largely considered as a black box, with few assumptions to be made about the noise [160]. In this context, the assumption of Pauli noise and the assumption that the noise of a circuit block are independent of the context. This means that the same gate causes the same noise at different positions in the circuit. Of course, these assumptions might not be fully satisfied in a real experiment. Instead of a description that is as complete as possible, we obtain information regarding errors that affect operators in the stabilizer elements, which we will discuss below.

3.7.2 Error Mitigation

Error mitigation is a form of post-processing in which one tries to infer the ideal result from the noisy result [151, 152]. The techniques of error mitigation use additional measurements to extract information about the noise, which can be partially removed from the outcome. They go beyond simply improving the measurement statistics by increasing the number of runs. However, they are interesting for NISQ computers because they do not require additional quantum resources. Prominent examples are zero-noise extrapolation [153] and readout error mitigation [154].

Furthermore, the method of [160] to model the noise above can be used as an error mitigation technique. The parameters of the error model are determined via a calibration measurement. It allows us to infer the ideal expected values from the noisy ones of the stabilizer elements. The results are comparable to readout error mitigation, while the method generates significantly less effort.

3.7.3 Error Correction

The topic of quantum error correction is vast and plays an important role. In this subsection, we briefly sketch the relevant concepts and refer interested readers to excellent introductions in [155, 156, 157]. Generally speaking, it is the extension of classical error correction codes to correct not only bit-flip but also phase-flip errors, and thus also general errors on a quantum system.

Quantum error correction codes encode k logical qubits into n physical qubits. Many codes can be described via the stabilizer of this code space, i.e., via a subgroup of the Pauli group whose elements leave the code words invariant. Small size examples are the nine-qubit Shor code [161], the seven-qubit Steane code [162], and the five-qubit code [163, 164]. A family of widely used codes is the surface code [165]. The layout of the qubits follows a lattice structure with the stabilizer generators acting locally, a fact that makes these codes a natural choice for hardware platforms with a matching architecture, e.g., those based on superconducting qubits.

The capacity of a code to correct errors is described by distance d, the minimum Hamming distance between two code words. The information about an error in the system is determined via the *syndrome measurements*. Here, one measures a set of observables that yield enough information to inform the correction operation. This measurement result is called a *syndrome*.

The concept of fault tolerance is crucial in quantum computing [155]. It is possible, with the help of quantum error correction codes and clever design of circuits, to perform arbitrarily long calculations despite the noisy operations. One can simply choose an arbitrarily large code, if it does not introduce too much noise due to the overhead of the additional operations, and the existing faults cannot propagate badly. It allows us to push the noise down to a desired level. The required quality of operations that achieves this scaling is called the threshold of the error correction scheme [166]. The additional complexity can be hidden in an abstract layer of the stack, e.g., when focusing on higher layers. It allows us to develop an ideal QC without having to consider the additional complexity of error correction at all times.

3.8 Verification and Benchmarking

Verification aims to ensure that software fulfills its requirements, e.g., that the output is correct under certain preconditions for given inputs. Similar to conventional non-deterministic software, the stochastic nature of most quantum algorithms poses a challenge to verification. That is, the same inputs can produce different results due to the intrinsic properties of quantum measurements but also due to the high level of noise on near-term hardware; see Sect. 3.2. A typical requirement that needs to be verified is that one obtains a high-quality solution, e.g., a result close to the desired result, with a sufficiently high probability. Here we focus on static verification, while what is sometimes also referred to as dynamical verification is covered in Sect. 3.5.4. The task of verification can be addressed from two sides. First, from a formal point of view, given "working" hardware we need a theoretical proof that the quantum algorithm is correct. Second, from a practical point of view, we need to ensure that the algorithm is implemented correctly in code for the classical and quantum parts of the program. Both parts need verification and the latter finally needs to be correctly translated into the executable circuit; see also Sect. 3.6. The verification of quantum algorithms gives rise to an interesting research question [167, 168, 126]: When quantum computers outperform conventional computers, how can we ensure that the algorithm is correct?

Benchmarking is the quantification of the performance of software and hardware. Because in the current state of QC the question is not yet how fast we can get a result but how good the results are that we get, benchmarking usually refers to assessing the quality of hardware components. So in contrast to conventional computer science, we do not yet compare different software or hardware with metrics like time to solution. In this context, benchmarks are standardized, technology-agnostic methods to evaluate quantum computers. The result of benchmarks are metrics for the performance of a specific device. They should be treated with caution, as they only cover single aspects of the machine, may struggle with the different hardware approaches (see also Sect. 3.2), and only measure the current state of the technology, not its future perspective. Also note that the score is affected by software, in particular the compilation. Any good benchmark should fulfill a number of requirements. It should be accepted by scientists and industry alike. The score of the benchmark is a number or a yes/no answer. The metric allows for a meaningful interpretation, which goes beyond that specific benchmark test. It should not give an advantage to one specific technology by construction, but the values can and will be better for some technologies than others, of course. The benchmarks should be well defined and easy to understand. They should be efficiently implementable, which poses a limitation on the information content in practice and therefore requires them to focus on specific aspects of the performance. They should be reproducible, which is a challenge given the non-deterministic character of quantum computers. Finally, they need to be scalable so they can be applied to small and larger quantum computers to enable tracking of the development progress.

The following quantities and methods are typically considered in the context of benchmarking.

- The fidelity of state preparation, single and two-qubit gates, and measurements. These numbers measure how close the implemented operation and the target operation are.

- Coherence times, which describe how long the coherence of a system, i.e., its ability to interfere, is conserved. In particular, the number of gate operations which can be performed in the coherence time indicates how long quantum computations can be.
- Cross-talk, e.g., how strong idle qubits are affected by gates acting on other qubits. Due to its non-local nature this noise can be difficult to handle.
- Hardware connectivity, i.e., how many qubits can directly interact.
- State and process tomography are methods that allow for a full characterization of a quantum state and a quantum operation, respectively [169, 170, 171].
- Randomized benchmarking [172] is a method to find the average gate fidelity of an important subset of gates, the so-called Clifford gates. It relies on the Gottesman–Knill theorem, which shows that circuits only consisting of such gates can be efficiently simulated on a conventional computer [10]. This then allows for the random insertion of gates into a circuit and efficient inversion of their net effect. Then the deviation from an identity operation is linked to the average fidelity of the gates. One advantage of this method is that the metric is not affected by state preparation and measurement errors.

In order to perform useful reproducible benchmarks, we need to define a suite of standard problems, ideally reflecting interesting target applications (like SPEC benchmarks [173] for different classical hardware) or, e.g., basic operations and algorithms (like LINPACK [174]). Existing benchmark suites for QC include SupermarQ [175] and Arline [130]. There are also benchmark suites tailored toward specific applications, e.g., fermionic quantum simulation [176]. In addition, for the special case of comparing quantum hardware and software, it might be helpful to further specify some constraints on how those problems should be solved as different approaches might not be comparable; e.g., hardcoding the solution is not a fair comparison. This can be tricky to achieve in practice as different assumptions or prior knowledge about the problem is often used in different solution approaches.

For comparison with classical computers, there exists a wide range of possible implementations: we can plug in a classical computer at almost any stage from the level of the original application problem, over a transformed formulation suitable for a quantum algorithm, to the actual operations for a specific hardware (at least for small problems or theoretical runtime considerations). And even on a classical computer, software can be more or less optimized, which influences its runtime by several orders of magnitudes (see, e.g., [177] for an example). So for actual benchmarking results, one needs to provide many additional details on all used implementations as well as on the hardware to actually allow an insightful comparison and interpretation. We further suggest defining separate benchmarking suites to address specific questions in the future:

- **Quantum supremacy**: These benchmarks compare the fastest implementation on a quantum computer with the fastest, elaborated, existing software for classical hardware for different key applications.

- **Near-term practicability**: These benchmarks compare the (estimated) costs of solutions for interesting algorithms (with input data from applications) for different quantum platforms and for classical hardware.
- **Performance and correctness**: These benchmarks assess the accuracy/quality of solutions obtained with different quantum hardware and software stacks for mathematical test problems with known solutions.

4 System Architecture and Implementation

In Sect. 3, we have described the conceptual workflow of quantum computers, from application to actual hardware. The end users are typically ultimately interested in solving their engineering problem, e.g., in simulating the airflow around an aircraft or in finding some optimal resource scheduling. To this end, we aim to construct a platform that allows end users to describe their domain-specific problem and find solutions to it while having to think about the underlying hardware as little as possible. This section describes the technical building blocks we use to construct such a platform. An illustration of the individual components and their connections is provided in Fig. 8.

We aim to construct this platform as domain-independently as possible. To guide our description of the individual components, however, we use an artificial example problem from the automotive domain using this platform. This example problem serves to highlight many of the considerations to be made when constructing a platform for quantum computing. While other use cases will require additional considerations, we believe that this example already suffices to illustrate the most pressing and general concerns platform engineers should consider for a wide swath of use cases.

For our example, consider the goal of developing a new driving function for autonomous vehicles. The engineers implementing this driving function want to evaluate whether it behaves safely in a number of specified driving scenarios. To this end, they specify sets of possible scenarios using traffic sequence charts (TSC) [178]. They then instantiate scenarios that conform to solutions for the given TSC problem and simulate the behavior of the implemented driving function in that scenario [179]. Moreover, they implement a software monitor that observes the simulation and reports unexpected behavior. This is also accompanied by a visualization of the simulation. The full sequence of steps of the simulation shall be automatized. To stay in the frame of a quantum software ecosystem, we further assume that the TSC problem shall be solved using quantum computing hardware. This can, for instance, be done by converting the TSC into a SAT formula and using Grover's algorithm to search for feasible solutions provided by a corresponding quantum oracle gate.

Note that, in an actual application, the TSC is converted into an Satisfiability Modulo Theory (SMT) formula instead of a SAT formula, where the former is a strictly more general model than the latter. There is, however, currently not

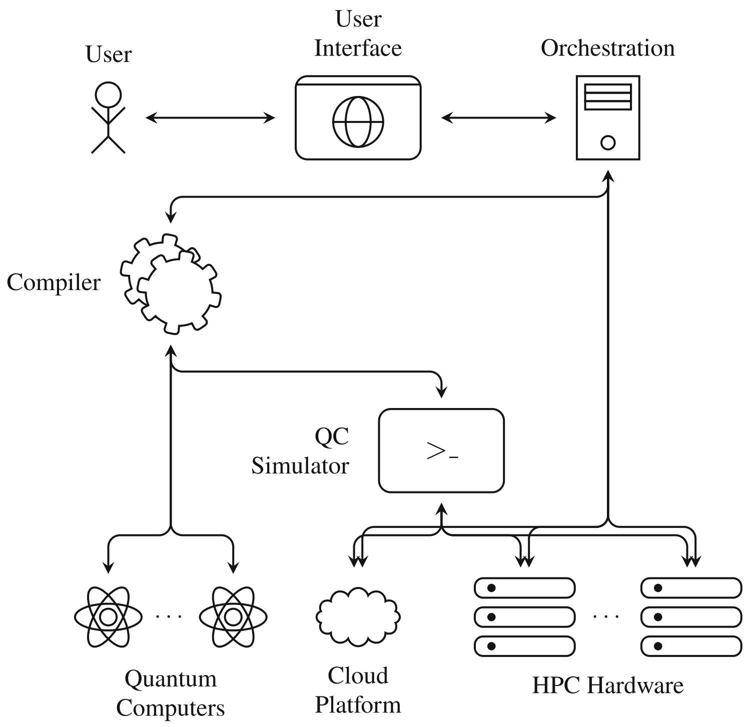

Fig. 8 A technical overview of the platform supporting quantum software developers

a straightforward way to generate solutions for an SMT formula using quantum circuits. Hence, for the sake of example, we assume that the engineers instead generate concrete scenarios via SAT formulas instead of SMT formulas.

In practice, the described simulation steps require heterogeneous hardware: the transformation from TSC to SAT and the extraction of a concrete scenario from a SAT solution can be executed on virtually any hardware without proprietary software. In contrast, finding the solution of the SAT problem and the execution of the simulation requires specialized software, namely SAT solvers, such as Z3 [180], and traffic simulation software, such as CARLA [181], respectively. Moreover, the visualization requires specialized hardware, e.g., graphics processing units (GPUs). As, in this running example, we assume that the engineer wants to find solutions to the SAT formula using some quantum circuit, we also interact with quantum devices.

In order to construct and execute their experiments, the engineers require some interface to the system. This interface provides the engineer with an integrated development environment (IDE) and, once the engineer is satisfied with their specification, passes the problem to some backend for execution. We describe the requirements for that interface in Sect. 4.1. Once the end user has specified the problem, the platform will have to schedule the use of the heterogeneous hardware systems described above. The major novelty of this platform lies in orchestrating the cooperation between classical computing hardware on the one hand, including, e.g., classical workstations, HPC resources, and GPUs, and between QPUs on the other hand. We describe the requirements for this orchestration in Sect. 4.2. The QPUs used during the execution may be implemented in actual hardware or it may be simulated using one of multiple quantum computing simulators. We have described the constraints faced in using existing hardware platforms in Sect. 3.2. The trade-offs to consider when using quantum simulators follow in Sect. 4.3.

4.1 User Interface

Quantum software developers require a straightforward interface for specifying their problems. In our example, the end user must be able to specify the described loop consisting of reading a TSC, calling external software, and performing computations on a well-suited QPU. The end user is not likely to be interested in the specifics of the underlying hardware but instead wants to have the choice of hardware handled by the platform during execution. In contrast, the interface should also cater to experts who are not interested in specifying domain-specific problems but are working on developing novel quantum algorithms. To this end, they require more direct access to the underlying hardware for, e.g., benchmarking.

The interface should allow the user to iterate rapidly on problem formulations e.g., the typical interface of HPC hardware. When using HPC hardware for solving a domain problem, the underlying algorithms and implementations are often mature and well tested. In contrast, when using quantum computing hardware, the underlying algorithms and implementations are constantly evolving and are often adapted to the domain problem at hand. Hence, the platform should allow the end user to rapidly iterate on the formulation of the domain-specific problem.

One approach to satisfy these requirements is allowing users to formulate their problems using a *service-oriented architecture*. In such an architecture, multiple independent software services collaborate to solve the specified problem. In our example above, users could specify one service each for the following tasks:

- Transform a given TSC into an SAT formula
- Construct a quantum circuit that solves this formula.
- Execute the quantum circuit to obtain a solution.
- Transform the solution into a concrete scenario.
- Simulate and monitor the scenario using CARLA [181], obtaining a visualization of the simulation.

The user needs to specify the software and hardware requirements for each service, e.g., that they require a QPU for the third service and CARLA with GPUs for the visualization of the fifth service. They do not necessarily have to implement all services themselves but can rely on other services that users of the platform have implemented and opted to share publicly. Finally, the user must specify the data flow between these services and ask the orchestration component to execute the composed service.

Letting end users define composable services and publishing them to other users has proven successful in the context of data analysis with Apache Nifi [182] and in the context of preliminary design of airplanes, jet fuels, electrical grids, ships, and other complex systems with RCE [183]. Moreover, a graphical user interface that allows users to graphically connect relevant services has been employed successfully for several decades in the field of data acquisition and analysis by LabVIEW [184].

4.2 Orchestration and Data Management

Once the problem has been specified and is given to the orchestration component for execution, that component has to reserve computation time on the initial required computing resource. Our running example requires some computation time on an off-the-shelf workstation which transforms the TSC into an SAT formula and subsequently transforms this formula into a quantum circuit. The orchestration component then has to reserve computation time on some QPU, either real hardware or simulated, to execute the quantum circuit. Once the execution of the quantum circuit has finished and resulted in a solution to the SAT problem, the orchestration component needs to reserve some computation time on an off-the-shelf workstation which transforms this solution into a scenario. Subsequently, the orchestration component needs to reserve computation time on an HPC resource equipped with GPUs to simulate the generated scenario, monitor the simulation, and visualize the simulation if necessary. Finally, the orchestration component needs to repeat the above steps until non-nominal behavior is observed during the simulation.

Our example shows that it is infeasible for the orchestration component to reserve all required computing resources prior to the execution of the initial service. The requirements for the QPU, the available gates, and number of qubits required for the execution of the quantum circuit only become available after the execution of the initial service. Hence, the orchestration component needs to be able to reserve computation time on the fly as results from earlier services become available.

Moreover, the orchestration component needs to take into account external requirements for the chosen computational resources. The visualization of the simulation in the final step of the computation described above may require large maps, textures, or other large data artifacts to visualize the scenario with the required fidelity. If these artifacts are only available to the visualization via a network connection with low bandwidth, the execution time of the complete computation will increase significantly. Hence, the orchestration component needs to be aware of data-intensive parts of the computation and the locality of the required data.

4.3 Use of QC Simulators

In the section Sect. 3.2, we have described the hardware platforms that are currently available for executing quantum circuits. All these platforms are costly, only available in low quantities, do not provide a large number of qubits, and produce noisy results. Although these problems are being addressed in the production of quantum computing hardware research, alternative solutions may tackle these issues.

A promising alternative is the use of quantum computing simulators that adopt classical hardware to simulate the execution of quantum circuits on actual hardware. Simulating such an execution requires significant computing power that is usually only provided by HPC systems. These systems are typically the same ones that execute the classical part of the computation job. Hence, any platform for the execution of quantum computing workloads using simulators must strike a balance between using the HPC resources it has available for the simulation of quantum circuits and using them for classical computation. Moreover, these HPC resources are rarely available for exclusive use by the quantum computing platform. Instead, the resources are also used for "classical" HPC applications. The owner of the resources has to balance their availability between the use by the platform and by the classical applications.

Although the results of the simulations produce data in the same order of magnitude as actual quantum computers (namely a few kilobytes or megabytes), they may offer additional diagnostic data which grows exponentially with the number of simulated qubits. If this data is made available to end users, the platform needs to provide data storage as well as bandwidth for transferring the data to the end user.

5 Conclusion

Quantum computing represents a paradigm shift in computational capabilities, with potential applications in various sectors. A key aspect to unlock its full potential is the establishment of a robust software ecosystem. This ecosystem not only provides the essential infrastructure for operating quantum devices but also serves as a bridge, enabling a broad spectrum of researchers, scientists, and industry experts to explore, use, and enhance the applications of these quantum systems.

Our chapter takes a research-driven approach toward constructing such an ecosystem. We have bifurcated our exploration into two key dimensions. Firstly, we present the conceptual design which encompasses considerations from computational paradigms, applications like quantum simulation, over device-optimized compiling to error handling, verification, and benchmarking. This underscores the theoretical foundation, taking into account the unique challenges and attributes of quantum computing. Secondly, we delve into the system architecture and implementation, focusing on aspects ranging from user interfaces to orchestration, data management, and the critical role of quantum computing simulators. The fusion of these two perspectives ensures a comprehensive understanding and a holistic approach to developing a quantum software ecosystem.

As we step into the future, it is imperative to emphasize that this endeavor is iterative. Practical evaluation and real-world implementation of the proposed ecosystem will undoubtedly reveal areas for improvement. The scientific approach allows the adaptation of the ecosystem, especially given the rapidly evolving quantum hardware landscape. Monitoring these advancements and ensuring flexibility in the response will be critical to remaining aligned with the dynamic nature of quantum computing. By doing so, we pave the way for maximizing the potential of quantum computing, fostering innovation, and moving the field forward.

Acknowledgments This research is part of the projects Algorithms for quantum computer development in hardware-software codesign (ALQU), https://qci.dlr.de/en/alqu, and Classical Integration of Quantum Computers (CLIQUE), https://qci.dlr.de/en/clique, which were made possible by the DLR Quantum Computing Initiative (QCI) and the German Federal Ministry for Economic Affairs and Climate Action (BMWK). Special thanks is due to IQM Quantum Computers and EleQtron for kindly making their respective quantum hardware chip designs available.

References

1. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997). https://doi.org/10.1137/S0097539795293172
2. Lloyd, S.: Universal quantum simulators. Science **273**(5278), 1073 (1996). https://doi.org/10.1126/science.273.5278.1073
3. Harrow, A.W., Hassidim, A., Lloyd, S.: Quantum algorithm for linear systems of equations. Phys. Rev. Lett. **103**, 150502 (2009). https://doi.org/10.1103/PhysRevLett.103.150502
4. Kadowaki, T., Nishimori, H.: Quantum annealing in the transverse Ising model. Phys. Rev. E **58**, 5355 (1998). https://doi.org/10.1103/PhysRevE.58.5355
5. Byrd, G.T., Ding, Y.: Quantum computing: progress and innovation. Computer **56**(01), 20 (2023). https://doi.org/10.1109/MC.2022.3217021
6. Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J.C., Barends, R., Biswas, R., Boixo, S., Brandao, F.G.S.L., Buell, D.A., Burkett, B., Chen, Y., Chen, Z., Chiaro, B., Collins, R., Courtney, W., Dunsworth, A., Farhi, E., Foxen, B., Fowler, A., Gidney, C., Giustina, M., Graff, R., Guerin, K., Habegger, S., Harrigan, M.P., Hartmann, M.J., Ho, A., Hoffmann, M., Huang, T., Humble, T.S., Isakov, S.V., Jeffrey, E., Jiang, Z., Kafri, D., Kechedzhi, K., Kelly, J., Klimov, P.V., Knysh, S., Korotkov, A., Kostritsa, F., Landhuis, D., Lindmark, M., Lucero, E., Lyakh, D., Mandrà, S., McClean, J.R., McEwen, M., Megrant, A., Mi, X., Michielsen, K., Mohseni, M., Mutus, J., Naaman, O., Neeley, M., Neill, C., Niu, M.Y., Ostby, E., Petukhov, A., Platt, J.C., Quintana, C., Rieffel, E.G., Roushan, P., Rubin, N.C., Sank, D., Satzinger, K.J., Smelyanskiy, V., Sung, K.J., Trevithick, M.D., Vainsencher, A., Villalonga, B., White, T., Yao, Z.J., Yeh, P., Zalcman, A., Neven, H., Martinis, J.M.: Quantum supremacy using a programmable superconducting processor. Nature **574**(7779), 505 (2019). https://doi.org/10.1038/s41586-019-1666-5

7. Preskill, J.: Quantum computing in the NISQ era and beyond. Quantum **2**, 79 (2018). https://doi.org/10.22331/q-2018-08-06-79
8. Temkin, M.. Investors bet on the technologically unproven field of quantum computing. https://pitchbook.com/news/articles/quantum-computing-venture-capital-funding. Accessed 04 Oct 2023
9. Cohen-Tannoudji, C., Diu, B., Laloe, F.: Quantum Mechanics. Textbook Physics, vol. 1, 1st edn. Wiley, Hoboken (1991)
10. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2000). https://doi.org/10.1017/CBO9780511976667
11. Schuhmacher, P.K: Decoherence as a resource for quantum information. Ph.D. Thesis, Universität des Saarlandes (2021). https://doi.org/10.22028/D291-35131
12. DiVincenzo, D.P.: The physical implementation of quantum computation. Fortschritte der Physik Progr. Phys. **48**(9–11), 771 (2000). https://doi.org/10.1002/1521-3978(200009)48:9/11<771::AID-PROP771>3.0.CO;2-E
13. Sleator, T., Weinfurter, H.: Realizable universal quantum logic gates. Phys. Rev. Lett. **74**, 4087 (1995). https://doi.org/10.1103/PhysRevLett.74.4087
14. Farhi, E., Goldstone, J., Gutmann, S., Sipser, M.: Quantum computation by adiabatic evolution. Preprint (2000). https://doi.org/10.48550/arXiv.quant-ph/0001106
15. Born, M., Fock, V.: Beweis des Adiabatensatzes. Zeitschrift für Physik **51**(3–4), 165–180 (1928). https://doi.org/10.1007/BF01343193
16. Van Dam, W., Mosca, M., Vazirani, U.: Proceedings 42nd IEEE Symposium on Foundations of Computer Science, pp. 279–287. IEEE, Piscataway (2001). https://doi.org/10.1109/SFCS.2001.959902
17. Aharonov, D., Van Dam, W., Kempe, J., Landau, Z., Lloyd, S., Regev, O.: Adiabatic quantum computation is equivalent to standard quantum computation. SIAM Rev. **50**(4), 755 (2008). https://doi.org/10.1137/080734479
18. Albash, T., Lidar, D.A.: Adiabatic quantum computation. Rev. Modern Phys. **90**(1), 015002 (2018). https://doi.org/10.1103/RevModPhys.90.015002
19. Pfeuty, P.: The one-dimensional Ising model with a transverse field. Ann. Phys. **57**, 79 (1970). https://doi.org/10.1016/0003-4916(70)90270-8
20. Fauseweh, B., Uhrig, G.S.: Multiparticle spectral properties in the transverse field Ising model by continuous unitary transformations. Phys. Rev. B **87**, 184406 (2013). https://doi.org/10.1103/PhysRevB.87.184406
21. Johnson, M.W., Amin, M.H., Gildert, S., Lanting, T., Hamze, F., Dickson, N., Harris, R., Berkley, A.J., Johansson, J., Bunyk, P., et al.: Quantum annealing with manufactured spins. Nature **473**(7346), 194 (2011). https://doi.org/10.1038/nature10012
22. Jünger, M., Lobe, E., Mutzel, P., Reinelt, G., Rendl, F., Rinaldi, G., Stollenwerk, T.: Quantum annealing versus digital computing: an experimental comparison. J. Exper. Algorithmics **26**, 1 (2021). https://doi.org/10.1145/3459606
23. McGeoch, C.C.: Theory versus practice in annealing-based quantum computing. Theoret. Comput. Sci. **816**, 169 (2020). https://doi.org/10.1016/j.tcs.2020.01.024
24. Raussendorf, R., Briegel, H.J.: A one-way quantum computer. Phys. Rev. Lett. **86**, 5188 (2001). https://doi.org/10.1103/PhysRevLett.86.5188
25. Knill, E., Laflamme, R., Milburn, G.J.: A scheme for efficient quantum computation with linear optics. Nature **409**(6816), 46 (2001). https://doi.org/10.1038/35051009
26. Browne, D.E., Rudolph, T.: Resource-efficient linear optical quantum computation. Phys. Rev. Lett. **95**, 010501 (2005). https://doi.org/10.1103/PhysRevLett.95.010501
27. Aharonov, Y., Davidovich, L., Zagury, N.: Quantum random walks. Phys. Rev. A **48**, 1687 (1993). https://doi.org/10.1103/PhysRevA.48.1687
28. Kempe, J.: Quantum random walks: An introductory overview. Contemp. Phys. **44**(4), 307 (2003). https://doi.org/10.1080/00107151031000110776

29. Childs, A.M., Cleve, R., Deotto, E., Farhi, E., Gutmann, S., Spielman, D.A.: Exponential algorithmic speedup by a quantum walk. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing, STOC '03, pp. 59–68. Association for Computing Machinery, New York (2003). https://doi.org/10.1145/780542.780552
30. Childs, A.M.: Universal computation by quantum walk. Phys. Rev. Lett. **102**, 180501 (2009). https://doi.org/10.1103/PhysRevLett.102.180501
31. Lovett, N.B., Cooper, S., Everitt, M., Trevers, M., Kendon, V.: Universal quantum computation using the discrete-time quantum walk. Phys. Rev. A **81**, 042330 (2010). https://doi.org/10.1103/PhysRevA.81.042330
32. Aharonov, D., Ambainis, A., Kempe, J., Vazirani, U.: Quantum walks on graphs. In: Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, STOC '01, pp. 50–59. Association for Computing Machinery, New York (2001). https://doi.org/10.1145/380752.380758
33. Farhi, E., Gutmann, S.: Quantum computation and decision trees. Phys. Rev. A **58**, 915 (1998). https://doi.org/10.1103/PhysRevA.58.915
34. Schuld, M., Sinayskiy, I., Petruccione, F.: Quantum walks on graphs representing the firing patterns of a quantum neural network. Phys. Rev. A **89**, 032333 (2014). https://doi.org/10.1103/PhysRevA.89.032333
35. Rebentrost, P., Mohseni, M., Lloyd, S.: Quantum support vector machine for big data classification. Phys. Rev. Lett. **113**, 130503 (2014). https://doi.org/10.1103/PhysRevLett.113.130503
36. Mohseni, M., Rebentrost, P., Lloyd, S., Aspuru-Guzik, A.: Environment-assisted quantum walks in photosynthetic energy transfer. J. Chem. Phys. **129**(17), 174106 (2008). https://doi.org/10.1063/1.3002335
37. Whitfield, J.D., Rodríguez-Rosario, C.A., Aspuru-Guzik, A.: Quantum stochastic walks: A generalization of classical random walks and quantum walks. Phys. Rev. A **81**, 022323 (2010). https://doi.org/10.1103/PhysRevA.81.022323
38. Govia, L.C.G., Taketani, B.G., Schuhmacher, P.K., Wilhelm, F.K.: Quantum simulation of a quantum stochastic walk. Quant. Sci. Technol. **2**(1), 015002 (2017). https://doi.org/10.1088/2058-9565/aa540b
39. Schuhmacher, P.K., Govia, L.C.G., Taketani, B.G., Wilhelm, F.K.: Quantum simulation of a discrete-time quantum stochastic walk. Europhys. Lett. **133**(5), 50003 (2021). https://doi.org/10.1209/0295-5075/133/50003
40. Turing, A.M.: On computable numbers, with an application to the entscheidungsproblem. Proc. Lond. Mathemat. Soc. **2**(1), 230 (1937). https://doi.org/10.1112/plms/s2-42.1.230
41. NobelPrize.org. Nobel Prize Outreach AB 2023. The nobel prize in physics 1956. https://www.nobelprize.org/prizes/physics/1956/summary/. Accessed 04 Oct 2023
42. Wilhelm, F.K., Steinwandt, R., Langenberg, B., Liebermann, P.J., Messinger, A., Schuhmacher, P.K., Misra-Spieldenner, A.: BSI Project Number 283 (2018). https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Quantencomputer/P283_QC_Studie-V_1_2.pdf?__blob=publicationFile&v=1
43. Häffner, H., Roos, C., Blatt, R.: Quantum computing with trapped ions. Phys. Rep. **469**(4), 155 (2008). https://doi.org/10.1016/j.physrep.2008.09.003
44. Monroe, C., Kim, J.: Scaling the ion trap quantum processor. Science **339**(6124), 1164 (2013). https://doi.org/10.1126/science.1231298
45. Brandl, M.F.: A Quantum von Neumann Architecture for Large-Scale Quantum Computing. Preprint (2017). https://doi.org/10.48550/arXiv.1702.02583
46. Clarke, J., Wilhelm, F.K.: Superconducting quantum bits. Nature **453**(7198), 1031 (2008). https://doi.org/10.1038/nature07128
47. Kane, B.E.: A silicon-based nuclear spin quantum computer. Nature **393**(6681), 133 (1998). https://doi.org/10.1038/30156
48. Heinzel, T.: Mesoscopic Electronics in Solid State Nanostructures. Physics Textbook, 2nd edn. Wiley, Hoboken (2007)

49. Hayashi, T., Fujisawa, T., Cheong, H.D., Jeong, Y.H., Hirayama, Y.: Coherent manipulation of electronic states in a double quantum dot. Phys. Rev. Lett. **91**, 226804 (2003). https://doi.org/10.1103/PhysRevLett.91.226804
50. Acín, A., Bloch, I., Buhrman, H., Calarco, T., Eichler, C., Eisert, J., Esteve, D., Gisin, N., Glaser, S.J., Jelezko, F., Kuhr, S., Lewenstein, M., Riedel, M.F., Schmidt, P.O., Thew, R., Wallraff, A., Walmsley, I., Wilhelm, F.K.: The quantum technologies roadmap: a European community view. New J. Phys. **20**(8), 080201 (2018). https://doi.org/10.1088/1367-2630/aad1ea
51. Josephson, B.D.: Possible new effects in superconductive tunnelling. Phys. Lett. **1**(7), 251 (1962). https://doi.org/10.1016/0031-9163(62)91369-0
52. Bladh, K., Duty, T., Gunnarsson, D., Delsing, P.: The single Cooper-pair box as a charge qubit. New J. Phys. **7**(1), 180 (2005). https://doi.org/10.1088/1367-2630/7/1/180
53. Vion, D., Aassime, A., Cottet, A., Joyez, P., Pothier, H., Urbina, C., Esteve, D., Devoret, M.H.: Manipulating the quantum state of an electrical circuit. Science **296**(5569), 886 (2002). https://doi.org/10.1126/science.1069372
54. Koch, J., Yu, T.M., Gambetta, J., Houck, A.A., Schuster, D.I., Majer, J., Blais, A., Devoret, M.H., Girvin, S.M., Schoelkopf, R.J.: Charge-insensitive qubit design derived from the Cooper pair box. Phys. Rev. A **76**, 042319 (2007). https://doi.org/10.1103/PhysRevA.76.042319
55. Mooij, J., Orlando, T., Levitov, L., Tian, L.,Van der Wal, C.H., Lloyd, S.: Josephson persistent-current qubit. Science **285**(5430), 1036 (1999). https://doi.org/10.1126/science.285.5430.1036
56. Van Der Wal, C.H., Ter Haar, A., Wilhelm, F., Schouten, R., Harmans, C., Orlando, T., Lloyd, S., Mooij, J.: Quantum superposition of macroscopic persistent-current states. Science **290**(5492), 773 (2000). https://doi.org/10.1126/science.290.5492.773
57. Pop, I.M., Geerlings, K., Catelani, G., Schoelkopf, R.J., Glazman, L.I., Devoret, M.H.: Coherent suppression of electromagnetic dissipation due to superconducting quasiparticles. Nature **508**(7496), 369 (2014). https://doi.org/10.1038/nature13017
58. Houck, A.A., Koch, J., Devoret, M.H., Girvin, S.M., Schoelkopf, R.J.: Life after charge noise: recent results with transmon qubits. Quant. Inf. Process. **8**(2), 105 (2009). https://doi.org/10.1007/s11128-009-0100-6
59. Clarke, J., Braginski, A.: The SQUID Handbook: Fundamentals and Technology of SQUIDs and SQUID Systems, vol. 1. Wiley, Hoboken (2004). https://doi.org/10.1002/3527603646
60. Lobe, E.: Combinatorial problems in programming quantum annealers. Ph.D. Thesis, Fakultät für Mathematik, Otto-von-Guericke-Universität Magdeburg (2022). https://doi.org/10.25673/89443
61. Motzoi, F., Gambetta, J.M., Rebentrost, P., Wilhelm, F.K.: Simple pulses for elimination of leakage in weakly nonlinear qubits. Phys. Rev. Lett. **103**, 110501 (2009). https://doi.org/10.1103/PhysRevLett.103.110501
62. Paul, W., Steinwedel, H.: Notizen: Ein neues Massenspektrometer ohne Magnetfeld. Zeitschrift für Naturforschung A **8**(7), 448 (1953). https://doi.org/10.1515/zna-1953-0710
63. Paul, W.: Electromagnetic traps for charged and neutral particles. Rev. Mod. Phys. **62**, 531 (1990). https://doi.org/10.1103/RevModPhys.62.531
64. Kielpinski, D., Monroe, C., Wineland, D.J.: Architecture for a large-scale ion-trap quantum computer. Nature **417**(6890), 709 (2002). https://doi.org/10.1038/nature00784
65. Monroe, C., Raussendorf, R., Ruthven, A., Brown, K.R., Maunz, P., Duan, L.M., Kim, J.: Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects. Phys. Rev. A **89**, 022317 (2014). https://doi.org/10.1103/PhysRevA.89.022317
66. Lekitsch, B., Weidt, S., Fowler, A.G., Mølmer, K., Devitt, S.J., Wunderlich, C., Hensinger, W.K.: Blueprint for a microwave trapped ion quantum computer. Sci. Adv. **3**(2), e1601540 (2017). https://doi.org/10.1126/sciadv.1601540
67. Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., Lloyd, S.: Quantum machine learning. Nature **549**(7671), 195 (2017). https://doi.org/10.1038/nature23474

68. Saggio, V., Asenbeck, B.E., Hamann, A., Strömberg, T., Schiansky, P., Dunjko, V., Friis, N., Harris, N.C., Hochberg, M., Englund, D., Wölk, S., Briegel, H.J., Walther, P.: Experimental quantum speed-up in reinforcement learning agents. Nature **591**(7849), 229 (2021). https://doi.org/10.1038/s41586-021-03242-7
69. Feynman, R.P., et al.: Simulating physics with computers. Int. J. Theor. Phys. **21**(6/7), 467–488 (2018)
70. Fauseweh, B., Schering, P., Hüdepohl, J., Uhrig, G.S.: Efficient algorithms for the dynamics of large and infinite classical central spin models. Phys. Rev. B **96**, 054415 (2017). https://doi.org/10.1103/PhysRevB.96.054415
71. Paeckel, S., Fauseweh, B., Osterkorn, A., Köhler, T., Manske, D., Manmana, S.R.: Detecting superconductivity out of equilibrium. Phys. Rev. B **101**, 180507 (2020). https://doi.org/10.1103/PhysRevB.101.180507
72. Schwarz, L., Fauseweh, B., Tsuji, N., Cheng, N., Bittner, N., Krull, H., Berciu, M., Uhrig, G.S., Schnyder, A.P., Kaiser, S., Manske, D.: Classification and characterization of nonequilibrium Higgs modes in unconventional superconductors. Nat. Commun. **11**(1), 287 (2020). https://doi.org/10.1038/s41467-019-13763-5
73. Fauseweh, B., Zhu, J.X.: Laser pulse driven control of charge and spin order in the two-dimensional Kondo lattice. Phys. Rev. B **102**, 165128 (2020). https://doi.org/10.1103/PhysRevB.102.165128
74. Peruzzo, A., McClean, J., Shadbolt, P., Yung, M.H., Zhou, X.Q., Love, P.J., Aspuru-Guzik, A., O'Brien, J.L.: A variational eigenvalue solver on a photonic quantum processor. Nat. Commun. **5**(1), 4213 (2014). https://doi.org/10.1038/ncomms5213
75. Fauseweh, B., Zhu, J.X.: Quantum computing Floquet energy spectra. Quantum **7**, 1063 (2023). https://doi.org/10.22331/q-2023-07-20-1063
76. Fauseweh, B., Zhu, J.X.: Digital quantum simulation of non-equilibrium quantum many-body systems. Quant. Inf. Process. **20**(4), 138 (2021). https://doi.org/10.1007/s11128-021-03079-z
77. Stollenwerk, T., O'Gorman, B., Venturelli, D., Mandra, S., Rodionova, O., Ng, H., Sridhar, B., Rieffel, E.G., Biswas, R.: Quantum annealing applied to de-conflicting optimal trajectories for air traffic management. IEEE Trans. Intell. Transport. Syst. **21**(1), 285 (2019). https://doi.org/10.1109/TITS.2019.2891235
78. Stollenwerk, T., Lobe, E., Jung, M.: International Workshop on Quantum Technology and Optimization Problems, pp. 99–110. Springer, Berrlin (2019). https://doi.org/10.1007/978-3-030-14082-3_9
79. Stollenwerk, T., Michaud, V., Lobe, E., Picard, M., Basermann, A., Botter, T.: Agile earth observation satellite scheduling with a quantum annealer. IEEE Trans. Aerosp. Electr. Syst. **57**(5), 3520 (2021). https://doi.org/10.1109/TAES.2021.3088490
80. Misra-Spieldenner, A., Bode, T., Schuhmacher, P.K., Stollenwerk, T., Bagrets, D., Wilhelm, F.K.: Mean-field approximate optimization algorithm. PRX Quantum **4**, 030335 (2023). https://doi.org/10.1103/PRXQuantum.4.030335
81. Jordan, S.: Quantum Algorithm Zoo. https://quantumalgorithmzoo.org/. Accessed 04 Oct 2023
82. Montanaro, A.: Quantum algorithms: an overview. npj Quant. Inf. **2**, 15023 (2016). https://doi.org/10.1038/npjqi.2015.23
83. Deutsch, D., Jozsa, R.: Rapid solution of problems by quantum computation. Proc. R. Soc. London. Ser. A Math. Phys. Sci. **439**(1907), 553 (1992). https://doi.org/10.1098/rspa.1992.0167
84. Bernstein, E., Vazirani, U.: Quantum complexity theory. SIAM J. Comput. **26**(5), 1411 (1997). https://doi.org/10.1137/S0097539796300921
85. Simon, D.R.: On the power of quantum computation. SIAM J. Comput. **26**(5), 1474 (1997). https://doi.org/10.1137/S0097539796298637

86. Tilly, J., Chen, H., Cao, S., Picozzi, D., Setia, K., Li, Y., Grant, E., Wossnig, L., Rungger, I., Booth, G.H., Tennyson, J.: The variational quantum eigensolver: A review of methods and best practices. Phys. Rep. **986**, 1 (2022). https://doi.org/10.1016/j.physrep.2022.08.003. The Variational Quantum Eigensolver: a review of methods and best practices
87. Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S.C., Endo, S., Fujii, K., McClean, J.R., Mitarai, K., Yuan, X., Cincio, L., Coles, P.J.: Variational quantum algorithms. Nat. Rev. Phys. **3**(9), 625 (2021). https://doi.org/10.1038/s42254-021-00348-9
88. McArdle, S., Jones, T., Endo, S., Li, Y., Benjamin, S.C., Yuan, X.: Variational ansatz-based quantum simulation of imaginary time evolution. npj Quant. Inf. **5**, 75 (2019). https://doi.org/10.1038/s41534-019-0187-2
89. Farhi, E., Goldstone, J., Gutmann, S.: A quantum approximate optimization algorithm. Preprint (2014). https://doi.org/10.48550/arXiv.1411.4028
90. Serrano, M.A., Perez-Castillo, R., Piattini, M.: Quantum Software Engineering. Springer Nature, Cham (2022). https://doi.org/10.1007/978-3-031-05324-5
91. Spoletini, P.: Towards quantum requirements engineering. In: 2023 IEEE 31st International Requirements Engineering Conference Workshops (REW), pp. 371–374. IEEE, Piscataway (2023). https://doi.org/10.1109/REW57809.2023.00072
92. Yue, T., Ali, S., Arcaini, P.: Towards quantum software requirements engineering. Preprint (2023). https://doi.org/10.48550/arXiv.2309.13358
93. ELEVATE (Enhanced probLEm solVing with quAntum compuTErs). https://www.dlr.de/sc/en/desktopdefault.aspx/tabid-18455/29433_read-77059/. Accessed 04 Oct 2023
94. DLR Quantum Computing Initiative – We shape the quantum computing ecosystem. https://qci.dlr.de/en/. Accessed 04 Oct 2023
95. Vandersypen, L.M.K., Steffen, M., Breyta, G., Yannoni, C.S., Sherwood, M.H., Chuang, I.L.: Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. Nature **414**(6866), 883 (2001). https://doi.org/10.1038/414883a
96. Amico, M., Saleem, Z.H., Kumph, M.: Experimental study of Shor's factoring algorithm using the IBM Q experience. Phys. Rev. A **100**, 012305 (2019). https://doi.org/10.1103/PhysRevA.100.012305
97. Skosana, U., Tame, M.: Demonstration of Shor's factoring algorithm for $N = 21$ on IBM quantum processors. Sci. Rep. **11**(1), 16599 (2021). https://doi.org/10.1038/s41598-021-95973-w
98. Zhang, X.M., Li, T., Yuan, X.: Quantum state preparation with optimal circuit depth: Implementations and applications. Phys. Rev. Lett. **129**, 230504 (2022). https://doi.org/10.1103/PhysRevLett.129.230504
99. Cirac, J.I., Blatt, R., Parkins, A.S., Zoller, P.: Preparation of Fock states by observation of quantum jumps in an ion trap. Phys. Rev. Lett. **70**, 762 (1993). https://doi.org/10.1103/PhysRevLett.70.762
100. Wunderlich, H., Wunderlich, C., Singer, K., Schmidt-Kaler, F.: Two-dimensional cluster-state preparation with linear ion traps. Phys. Rev. A **79**, 052324 (2009). https://doi.org/10.1103/PhysRevA.79.052324
101. Berwald, J., Chancellor, N., Dridi, R.: Understanding domain-wall encoding theoretically and experimentally. Philosoph. Trans. Roy. Soc. A: Math. Phys. Eng. Sci. **381**(2241), 20210410 (2023). https://doi.org/10.1098/rsta.2021.0410
102. Chancellor, N.: Domain wall encoding of discrete variables for quantum annealing and QAOA. Quant. Sci. Technol. **4**(4), 045004 (2019). https://doi.org/10.1088/2058-9565/ab33c2
103. Lloyd, S., Schuld, M., Ijaz, A., Izaac, J., Killoran, N.: Quantum embeddings for machine learning. Preprint (2020). https://doi.org/10.48550/arXiv.2001.03622
104. Cross, A.W., Bishop, L.S., Smolin, J.A., Gambetta, J.M.: Open quantum assembly language. Preprint (2017). https://doi.org/10.48550/arXiv.1707.03429
105. Cirq developers. Cirq (2023). https://doi.org/10.5281/zenodo.8161252. Full list of authors: http://github.com/quantumlib/Cirq/graphs/contributors
106. Qiskit contributors. Qiskit: An open-source framework for quantum computing (2023). https://doi.org/10.5281/zenodo.2573505. https://qiskit.org/

107. Svore, K., Geller, A., Troyer, M., Azariah, J., Granade, C., Heim, B., Kliuchnikov, V., Mykhailova, M., Paz, A., Roetteler, M.: Q#: Enabling scalable quantum computing and development with a high-level DSL. In: Proceedings of the Real World Domain Specific Languages Workshop 2018, RWDSL2018. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3183895.3183901
108. Bichsel, B., Baader, M., Gehr, T., Vechev, M.: Silq: a high-level quantum language with safe uncomputation and intuitive semantics. In: Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2020, pp. 286–300. Association for Computing Machinery, New York (2020). https://doi.org/10.1145/3385412.3386007
109. Foundation, E.: Qrisp (2023). https://www.qrisp.eu/. Accessed 21 Nov 2023
110. QIR Alliance: QIR Specification (2021). https://github.com/qir-alliance/qir-spec. Accessed 04 Oct 2023
111. Peduri, A., Bhat, S., Grosser, T.: QSSA: an SSA-based IR for quantum computing. In: Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction, CC 2022, pp. 2–14. Association for Computing Machinery, New York (2022). https://doi.org/10.1145/3497776.3517772
112. Pérez-Castillo, R., Piattini, M.: Design of classical-quantum systems with UML. Computing **104**(11), 2375 (2022). https://doi.org/10.1007/s00607-022-01091-4
113. Usaola, M.P.: In: Short Papers Proceedings of the 1st International Workshop on the QuANtum SoftWare Engineering & Programming, Talavera de la Reina, Spain, February 11–12, 2020, CEUR Workshop Proceedings. Piattini, M., Peterssen, G., Pérez-Castillo, R., Hevia, J.L., Serrano, M.A. (eds.) CEUR-WS.org, vol. 2561, pp. 57–63 (2020). https://ceur-ws.org/Vol-2561/paper6.pdf
114. García de la Barrera, A., García-Rodríguez de Guzmán, I., Polo, M., Piattini, M.: Quantum software testing: State of the art. J. Softw. Evolut. Process **35**(4), e2419 (2023). https://doi.org/10.1002/smr.2419
115. Miranskyy, A., Zhang, L., Doliskani, J.: On Testing and Debugging Quantum Software. Preprint (2021). https://doi.org/10.48550/arXiv.2103.09172
116. Wootters, W.K., Zurek, W.H.: A single quantum cannot be cloned. Nature **299**(5886), 802 (1982). https://doi.org/10.1038/299802a0
117. Li, G., Zhou, L., Yu, N., Ding, Y., Ying, M., Xie, Y.: Proq: Projection-based Runtime Assertions for Debugging on a Quantum Computer. Preprint (2020). https://doi.org/10.48550/arXiv.1911.12855
118. Liu, J., Byrd, G.T., Zhou, H.: Quantum Circuits for Dynamic Runtime Assertions in Quantum Computation. In: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '20, pp. 1017–1030. Association for Computing Machinery, New York (2020). https://doi.org/10.1145/3373376.3378488
119. Campos, J., Souto, A.: QBugs: A Collection of Reproducible Bugs in Quantum Algorithms and a Supporting Infrastructure to Enable Controlled Quantum Software Testing and Debugging Experiments. Preprint (2021). https://doi.org/10.48550/arXiv.2103.16968
120. Zhao, P., Zhao, J., Miao, Z., Lan, S.: Bugs4Q: A Benchmark of Real Bugs for Quantum Programs. Preprint (2021). https://doi.org/10.48550/arXiv.2108.09744
121. Honarvar, S., Mousavi, M.R., Nagarajan, R.: Property-based Testing of Quantum Programs in Q#. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW'20, pp. 430–435. Association for Computing Machinery, New York (2020). https://doi.org/10.1145/3387940.3391459
122. Gottesman, D.: Theory of fault-tolerant quantum computation. Phys. Rev. A **57**, 127 (1998). https://doi.org/10.1103/PhysRevA.57.127
123. Preskill, J.: Lecture Notes for Physics 229: Quantum Information and Computation. California Institution of Technology, Pasadena (1998)
124. Herr, D., Nori, F., Devitt, S.J.: Optimization of lattice surgery is NP-hard. npj Quant. Inf. **3**(1), 35 (2017). https://doi.org/10.1038/s41534-017-0035-1

125. Botea, A., Kishimoto, A., Marinescu, R.: In: Proceedings of the Eleventh International Symposium on Combinatorial Search (SoCS2018), vol. 9, pp. 138–142 (2018). https://doi.org/10.1609/socs.v9i1.18463
126. Amy, M., Azimzadeh, P., Mosca, M.: On the controlled-NOT complexity of controlled-NOT-phase circuits. Quant. Sci. Technol. **4**(1), 015002 (2018). https://doi.org/10.1088/2058-9565/aad8ca
127. Amy, M., Maslov, D., Mosca, M., Roetteler, M.: A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. IEEE Trans. Comput.-Aided Design Integr. Circ. Syst. **32**(6), 818 (2013). https://doi.org/10.1109/TCAD.2013.2244643
128. Schneider, S., Burgholzer, L., Wille, R.: In: Proceedings of the 28th Asia and South Pacific Design Automation Conference, ASPDAC '23, pp. 190–195 . Association for Computing Machinery, New York (2023). https://doi.org/10.1145/3566097.3567929
129. Nam, Y., Ross, N.J., Su, Y., Childs, A.M., Maslov, D.: Automated optimization of large quantum circuits with continuous parameters. npj Quant. Inf. **4**(1), 23 (2018). https://doi.org/10.1038/s41534-018-0072-4
130. Kharkov, Y., Ivanova, A., Mikhantiev, E., Kotelnikov, A.: Arline Benchmarks: Automated Benchmarking Platform for Quantum Compilers. Preprint (2022). https://doi.org/10.48550/arXiv.2202.14025
131. Khalate, P., Wu, X.C., Premaratne, S., Hogaboam, J., Holmes, A., Schmitz, A., Guerreschi, G.G. , Zou, X., Matsuura, A.Y.: An LLVM-based C++ Compiler Toolchain for Variational Hybrid Quantum-Classical Algorithms and Quantum Accelerators. Preprint (2022). https://doi.org/10.48550/arXiv.2202.11142
132. Epping, M.: Hybrid simplification rules for boundaries of quantum circuits. Preprint (2022). https://doi.org/10.48550/arXiv.2206.03036
133. NVIDIA, Vingelmann, P., Fitzek, F.H.: CUDA (2020). https://developer.nvidia.com/cuda-toolkit
134. Briegel, H.J., Browne, D.E., Dür, W., Raussendorf, R., Van den Nest, M.: Measurement-based quantum computation. Nat. Phys. **5**(1), 19 (2009). https://doi.org/10.1038/nphys1157
135. Lippert, T., Michielsen, K.: In: NIC Symposium 2022: Proceedings, Publication Series of the John von Neumann Institute for Computing (NIC) NIC Series, vol. 51, pp. 3 – 23. NIC Symposium 2022, Jülich, Germany, 29 Sep 2022–30 Sep 2022. Forschungszentrum Jülich GmbH Zentralbibliothek, Verlag, Jülich (2022). https://juser.fz-juelich.de/record/917067
136. HPCQC. Where quantum accelerates the future of supercomputing. https://www.hpcqc.org/. Accessed 04 Oct 2023
137. Janzing, D., Wocjan, P., Beth, T.: Identity check is QMA-complete. Preprint (2003). https://doi.org/10.48550/arXiv.quant-ph/0305050
138. Viamontes, G.F., Markov, I.L., Hayes, J.P.: In: 2007 IEEE/ACM International Conference on Computer-Aided Design, pp. 69–74 (2007). https://doi.org/10.1109/ICCAD.2007.4397246
139. Burgholzer, L., Wille, R.: In: 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC), IEEE, pp. 127–132. IEEE Press, Piscataway (2020). https://doi.org/10.1109/ASP-DAC47756.2020.9045153
140. Burgholzer, L., Wille, R.: QCEC: A JKQ tool for quantum circuit equivalence checking. Softw. Impacts **7**, 100051 (2021). https://doi.org/10.1016/j.simpa.2020.100051. https://www.sciencedirect.com/science/article/pii/S2665963820300427
141. D-Wave. Ocean. https://github.com/dwavesystems/dwave-ocean-sdk
142. DLR-SC. `quark` (2023). https://gitlab.com/quantum-computing-software/quark
143. Lobe, E., Lutz, A.: Minor Embedding in Broken Chimera and Derived Graphs is NP-complete. In: Theoretical Computer Science 989 (2024). https://doi.org/10.1016/j.tcs.2023.114369
144. Cai, J., W.G. Macready, Roy, A.: A practical heuristic for finding graph minors. Preprint (2014). https://doi.org/10.48550/arXiv.1406.2741
145. Lobe, E., Schürmann, L., Stollenwerk, T.: Embedding of complete graphs in broken Chimera graphs. Quant. Inf. Process. **20**(7), 1 (2021). https://doi.org/10.1007/s11128-021-03168-z
146. Lobe, E., Kaibel, V.: Optimal sufficient requirements on the embedded Ising problem in polynomial time. Quant. Inf. Process. **22**(305), 1 (2023). https://doi.org/10.1007/s11128-023-04058-2

147. Lidar, D.A.: Review of Decoherence-Free Subspaces, Noiseless Subsystems, and Dynamical Decoupling, pp. 295–354. Wiley, Hoboken (2014). https://doi.org/10.1002/9781118742631.ch11
148. Viola, L., Lloyd, S.: Dynamical suppression of decoherence in two-state quantum systems. Phys. Rev. A **58**, 2733 (1998). https://doi.org/10.1103/PhysRevA.58.2733
149. Lidar, D.A., Birgitta Whaley, K.: Decoherence-Free Subspaces and Subsystems, pp. 83–120. Springer, Berlin (2003). https://doi.org/10.1007/3-540-44874-8_5
150. Mueller, T., Stollenwerk, T., Headley, D., Epping, M., Wilhelm, F.K.: Coherent and non-unitary errors in ZZ-generated gates. Preprint (2023). https://doi.org/10.48550/arXiv.2304.14212
151. Cai, Z., Babbush, R., Benjamin, S.C., Endo, S., Huggins, W.J., Li, Y., McClean, J.R., O'Brien, T.E.: Quantum Error Mitigation. Preprint (2023). https://doi.org/10.48550/arXiv.2210.00921
152. Endo, S.: Hybrid quantum-classical algorithms and error mitigation. Ph.D. Thesis, University of Oxford (2019). https://ora.ox.ac.uk/objects/uuid:6733c0f6-1b19-4d12-a899-18946aa5df85
153. Temme, K., Bravyi, S., Gambetta, J.M.: Error mitigation for short-depth quantum circuits. Phys. Rev. Lett. **119**, 180509 (2017). https://doi.org/10.1103/PhysRevLett.119.180509
154. Beisel, M., Barzen, J., Leymann, F., Truger, F., Weder, B., Yussupov, V.: Configurable readout error mitigation in quantum workflows. Electronics **11**(19), 2983 (2022). https://doi.org/10.3390/electronics11192983
155. Gottesman, D.: An Introduction to Quantum Error Correction and Fault-Tolerant Quantum Computation. Preprint (2009). https://doi.org/10.48550/arXiv.0904.2557
156. Bacon, D.: Introduction to Quantum Error Correction, Chap. 2, pp. 46–77. Cambridge University Press, Cambridge (2013). https://doi.org/10.1017/CBO9781139034807.004
157. Roffe, J.: Quantum error correction: an introductory guide. Contemp. Phys. **60**(3), 226 (2019). https://doi.org/10.1080/00107514.2019.1667078
158. Mohseni, M., Rezakhani, A.T., Lidar, D.A.: Quantum-process tomography: Resource analysis of different strategies. Phys. Rev. A **77**, 032322 (2008). https://doi.org/10.1103/PhysRevA.77.032322
159. Flammia, S.T., Wallman, J.J.: Efficient estimation of pauli channels. ACM Trans. Quant. Comput. **1**(1), 1 (2020). https://doi.org/10.1145/3408039
160. Wimmer, C., Szangolies, J., Epping, M.: Calibration of Syndrome Measurements in a Single Experiment. Preprint (2023). https://doi.org/10.48550/arXiv.2305.03004
161. Shor, P.W.: Scheme for reducing decoherence in quantum computer memory. Phys. Rev. A **52**, R2493 (1995). https://doi.org/10.1103/PhysRevA.52.R2493
162. Steane, A.: Multiple-particle interference and quantum error correction. Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. **452**(1954), 2551 (1996). https://doi.org/10.1098/rspa.1996.0136
163. Laflamme, R., Miquel, C., J.P. Paz, Zurek, W.H.: Perfect Quantum Error Correction Code. Preprint (1996). https://doi.org/10.48550/arXiv.quant-ph/9602019
164. Bennett, C.H., DiVincenzo, D.P., Smolin, J.A., Wootters, W.K.: Mixed-state entanglement and quantum error correction. Phys. Rev. A **54**, 3824 (1996). https://doi.org/10.1103/PhysRevA.54.3824
165. Kitaev, A.Y.: Quantum computations: algorithms and error correction. Russ. Math. Surv. **52**(6), 1191 (1997). https://doi.org/10.1070/RM1997v052n06ABEH002155
166. Shor, P.: In: Proceedings of 37th Conference on Foundations of Computer Science, pp. 56–65 (1996). https://doi.org/10.1109/SFCS.1996.548464
167. Gheorghiu, A., Kapourniotis, T., Kashefi, E.: Verification of quantum computation: an overview of existing approaches. Theory Comput. Syst, **63**(4), 715 (2019). https://doi.org/10.1007/s00224-018-9872-3
168. Wang, S.A., Lu, C.Y., Tsai, I.M., Kuo, S.Y.: An XQDD-based verification method for quantum circuits. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **E91-A**(2), 584–594 (2008). https://doi.org/10.1093/ietfec/e91-a.2.584
169. D'Ariano, G.M., Paris, M.G.A., Sacchi, M.F.: Quantum Tomography. Preprint (2003). https://doi.org/10.48550/arXiv.quant-ph/0302028

170. D'Ariano, G.M., Maccone, L., Presti, P.L.: Quantum calibration of measurement instrumentation. Phys. Rev. Lett. **93**, 250407 (2004). https://doi.org/10.1103/PhysRevLett.93.250407
171. Artiles, L.M., Gill, R.D., Gută, M.I.: An invitation to quantum tomography. J. Roy. Stat. Soc. Ser. B (Statist. Methodol.) **67**(1), 109 (2005). https://doi.org/10.1111/j.1467-9868.2005.00491.x
172. Gaebler, J.P., Meier, A.M., Tan, T.R., Bowler, R., Lin, Y., Hanneke, D., Jost, J.D., Home, J.P., Knill, E., Leibfried, D., Wineland, D.J.: Randomized benchmarking of multiqubit gates. Phys. Rev. Lett. **108**, 260503 (2012). https://doi.org/10.1103/PhysRevLett.108.260503
173. SPEC (Standard Performance Evaluation Corporation). SPEC benchmark and tools. https://spec.org/benchmarks.html. Accessed 04 Oct 2023
174. Dongarra, J.J., Luszczek, P., Petitet, A.: The LINPACK Benchmark: past, present and future. Concurr. Comput. Pract. Exper. **15**(9), 803 (2003). https://doi.org/10.1002/cpe.728
175. Tomesh, T., Gokhale, P., Omole, V., Ravi, G.S., Smith, K.N., Viszlai, J., Wu, X.C., Hardavellas, N., Martonosi, M.R., Chong, F.T.: SupermarQ: A scalable quantum benchmark suite. Preprint (2022). https://doi.org/10.48550/arXiv.2202.11045
176. Dallaire-Demers, P.L., Stechly, M., Gonthier, J.F., Bashige, N.T., Romero, J., Cao, Y.: An application benchmark for fermionic quantum simulations. Preprint (2020). https://doi.org/10.48550/arXiv.2003.01862
177. Röhrig-Zöllner, M., Thies, J., Basermann, A.: Performance of the low-rank TT-SVD for large dense tensors on modern MultiCore CPUs. SIAM J. Sci. Comput. **44**(4), C287 (2022). https://doi.org/10.1137/21m1395545
178. Damm, W., Möhlmann, E., Peikenkamp, T., Rakow, A.: LNCS, pp. 182–205. Springer, Berlin (2018). https://doi.org/10.1007/978-3-319-95246-8_11
179. Kröger, Scheidegger, Becker, Deublein, Fehlberg, Galassi, Hohl, Koester, Zanella: Autonomes fahren. ein treiber zukünftiger mobilität (2022). https://doi.org/10.5281/ZENODO.5907154
180. de Moura, L., Bjørner, N.: TACAS, pp. 337–340. Springer, Berlin (2008). https://doi.org/10.1007/978-3-540-78800-3_24
181. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: In: Proceedings of the 1st Annual Conference on Robot Learning, Proceedings of Machine Learning Research. Levine, S., Vanhoucke, V., Goldberg, K. (Eds.) , vol. 78, pp. 1–16. PMLR (2017). https://proceedings.mlr.press/v78/dosovitskiy17a.html
182. Apache Software Foundation, Cloudera, Hortonworks. Apache Nifi. https://nifi.apache.org/. Accessed 04 Oct 2023
183. Boden, B., Flink, J., Först, N., Mischke, R., Schaffert, K., Weinert, A., Wohlan, A., Schreiber, A.: RCE: An integration environment for engineering and science. SoftwareX **15**, 100759 (2021). https://doi.org/10.1016/j.softx.2021.100759
184. Texas Instruments. Laboratory Virtual Instrument Engineering Workbench (LabVIEW). https://www.ni.com/labview. Accessed 04 Oct 2023