Simulation of Quantum Computers: Review and Acceleration Opportunities

Published: 19 November 2025
Online AM: 11 September 2025
Accepted: 03 August 2025
Revised: 10 June 2025
Received: 14 October 2024

REVIEW-ARTICLE

Simulation of Quantum Computers: Review and Acceleration Opportunities

ALESSIO CICERO, Chalmers University of Technology, Gothenburg, Vastra Gotaland, Sweden

MOHAMMAD ALI MALEKI, Chalmers University of Technology, Gothenburg, Vastra Gotaland, Sweden

MUHAMMAD WAQAR AZHAR

ANTON FRISK KOCKUM, Chalmers University of Technology, Gothenburg, Vastra Gotaland, Sweden

PEDRO TRANCOSO, Chalmers University of Technology, Gothenburg, Vastra Gotaland, Sweden

Open Access Support provided by:

Chalmers University of Technology

Simulation of Quantum Computers: Review and Acceleration Opportunities

ALESSIO CICERO, Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, Goteborg, Sweden
MOHAMMAD ALI MALEKI, Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, Goteborg, Sweden
MUHAMMAD WAQAR AZHAR, Zeropoint Technologies AB, Goteborg, Sweden
ANTON FRISK KOCKUM, Department of Microtechnology and Nanoscience, Chalmers University of Technology, Goteborg, Sweden
PEDRO TRANCOSO, Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, Goteborg, Sweden

Quantum computing has the potential to revolutionise multiple fields by solving complex problems that cannot be solved in reasonable time with current classical computers. Nevertheless, the development of quantum computers is still in its early stages and the available systems have still very limited resources. As such, currently, the most practical way to develop and test quantum algorithms is to use classical simulators of quantum computers. In addition, the development of new quantum computers and their components also depends on simulations.

Given the characteristics of a quantum computer, their simulation is a very demanding application in terms of both computation and memory. As such, simulations do not scale well in current classical systems. Thus different optimisation and approximation techniques need to be applied at different levels.

This review provides an overview of the components of a quantum computer, the levels at which these components and the whole quantum computer can be simulated, and an in-depth analysis of different state-of-the-art acceleration approaches. Besides the optimisations that can be performed at the algorithmic level, this review presents the most promising hardware-aware optimisations and future directions that can be explored for improving the performance and scalability of the simulations.

CCS Concepts: • **Hardware** → *Quantum computation*; • **Computing methodologies** → **Simulation types and techniques;**

We acknowledge support from the Swedish Foundation for Strategic Research (grant number FUS21-0063), the Horizon Europe programme HORIZON-CL4-2022-QUANTUM-01-SGA via the project 101113946 OpenSuperQPlus100, and from the Knut and Alice Wallenberg Foundation through the Wallenberg Centre for Quantum Technology (WACQT). AFK is also supported by the Swedish Research Council (grant number 2019-03696) and the Swedish Foundation for Strategic Research (grant number FFL21-0279).

Authors' Contact Information: Alessio Cicero, Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, Goteborg, Sweden; e-mail: alessio.cicero@chalmers.se; Mohammad Ali Maleki, Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, Goteborg, Sweden; e-mail: mohammad.ali.maleki@chalmers.se; Muhammad Waqar Azhar, Zeropoint Technologies AB, Goteborg, Sweden; e-mail: waqar.azhar@zptcorp.com; Anton Frisk Kockum, Department of Microtechnology and Nanoscience, Chalmers University of Technology, Goteborg, Sweden; e-mail: anton.frisk.kockum@chalmers.se; Pedro Trancoso, Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, Goteborg, Sweden; e-mail: ppedro@chalmers.se.

ACM 2643-6817/2025/11-ART3
https://doi.org/10.1145/3762672

Additional Key Words and Phrases: Quantum computing, computer simulation, hardware acceleration, CPU, GPU, FPGA

ACM Reference Format:
Alessio Cicero, Mohammad Ali Maleki, Muhammad Waqar Azhar, Anton Frisk Kockum, and Pedro Trancoso. 2025. Simulation of Quantum Computers: Review and Acceleration Opportunities. *ACM Trans. Quantum Comput.* 7, 1, Article 3 (November 2025), 35 pages. https://doi.org/10.1145/3762672

1 Introduction

As we try to solve more and more complex problems such as developing new chemical compounds [139] or evaluating the physical properties of new materials [11], the demand for computational resources continues to grow. Certain problem cases are so complex that no existing computer system can solve them within a reasonable time. For such cases, *Quantum Computing* [143] is a promising emerging computing paradigm that can provide solutions to these problems [11, 24, 25, 38, 139, 141, 157, 191].

As quantum computing is still in its early stages of development, real machines are scarce and the existing ones have limited compute resources. Although several quantum computing experiments have shown promising results and quantum computers are being scaled up to hundreds of qubits [5, 7, 17, 103, 131], full quantum advantage has yet to be achieved.

In order for algorithms to be executed in a quantum computer, they are represented as quantum circuits. The ability of quantum computers to solve increasingly complex problems is limited by the maximum size of the executable quantum circuit. This size is mainly bounded by two factors: (1) the number of available qubits (the fundamental unit of a quantum circuit) and (2) the circuit depth, which refers to the number of distinct timesteps at which quantum gates are applied [143]. In current quantum computer implementations, not only are qubits few but also the time they are stable (i.e., their value remains reliable) is limited. This stable time is limited by the qubit implementation technology and also their sensitivity to fluctuations in the environment (e.g., magnetic fields and temperature).

Quantum circuit execution is just one element of the *Quantum Computing Stack* [9]. Other elements include mapping an algorithm to a quantum circuit, generating pulses to execute gates on qubits, and reliably reading out the results. To develop and test the various parts of the quantum computing stack, accurate and fast simulation tools are essential. Similarly to classical computers, different simulation tools are needed at various design and verification steps, each with a specific focus and level of detail. A generic quantum computer system is composed of different parts, and the development of each part can be assisted by a different type of simulation.

This review work focuses in particular on the simulation of the execution of quantum circuits. Simulations of quantum circuit execution can be performed at different levels, from simulating the behaviour of the quantum hardware platform [91, 158], to simulating the interaction between a few qubits and a coupler forming a gate [31, 73, 155], and simulating entire circuits [14, 96, 100, 158, 173, 183, 205].

While several simulators have recently been deployed and made publicly available, the need to simulate more complex algorithms, circuit configurations, or more accurately model qubit behaviour leads to an exponential increase in computational and memory demand for the simulations. Even on a supercomputer such as Summit [57], we can only do algorithmic simulations of a quantum circuit up to 47 qubits, which requires 2.8PB of memory [198]. Scaling up the simulations without increasing exponentially the amount of necessary resources, requires optimisations or approximations at different levels. As such, several classical computer techniques, such as data compression or optimised parallel execution, have been applied to reduce the memory requirements and accelerate the computations [12, 130].

Several review works on classical simulation of quantum computers are available, but their focus is different from the acceleration of the simulation. Some works focus on the state-of-the-art numerical methods, such as the work by Xu et al. [199] which gives a general overview, and Jones et al. [97] which focuses on the full-state simulation techniques. Other works focus both on reporting the state of the art for simulators as well as giving a high-level overview of the acceleration approach as in the work by Young et al. [202]. The work by Heng et al. [84] focus on presenting optimisations for GPU execution, while the review by Jamadagni et al. [93] focuses on setting up a benchmark for multiple simulators available, and comparing their performance. In contrast to the previous works, our work addresses the topic in a different way, with a goal of focusing on the possible hardware-aware acceleration techniques. It provides up-to-date details about the available simulator approaches, tools and techniques, and focuses on the possible optimisations for a broader selection of hardware platforms.

First, this review work presents an overview of the different types of simulators for the execution of quantum circuits. The simulators are organised into different categories, to help navigate the landscape of the available tools. Second, it provides an overview of acceleration techniques described in the state of the art to improve the performance of the simulations. Having a global overview of the existing proposed solutions, we will infer what the most promising trends are and speculate on the future directions for the acceleration of the simulation of quantum computers. In this context, different works propose different approaches to optimise the simulation on different hardware platforms, such as **Central Processing Unit** (**CPU**), **Graphics Processing Unit** (**GPU**), **Field Programmable Gate Array** (**FPGA**), or more complex setups such as a hybrid CPU and **Quantum Processing Unit** (**QPU**) core. A summary of the hardware-aware techniques observed in the analysed works is presented and this is used as a basis to extrapolate what the future directions in hardware-aware acceleration for quantum computer simulation should be.

The organisation of this survey is as follows. Section 2 introduces the essential concepts in quantum computing necessary for understanding the rest of the sections. In Section 3, we give an overview of the various parts of a quantum computer. Section 4 covers simulations at different levels and provides an overview of the available simulators for each of them, also introducing the main bottlenecks for scaling up the number of simulated qubits. Section 5 details the most common hardware platforms used for simulation. The following sections cover different acceleration methods, sorted by platform: CPU in Section 6, GPU in Section 7, and FPGA in Section 8. A summary and indication of future directions for hardware-aware optimisation is presented in Section 9 and Section 10 concludes this work.

2 Background

This section presents a quick overview and introduction to the basic quantum computer concepts.

2.1 Qubits

The fundamental computational element for the quantum computer is the qubit (quantum bit). Differently from normal bits, qubits can be in a state different from just $|0\rangle$ and $|1\rangle$: they can form a linear combinations of states, usually called a superposition:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix}. \tag{1}$$

It is possible to visualise the state of the qubit on the Bloch sphere in Figure 1 using the equivalences [143]:

$$\alpha = \cos\frac{\theta}{2},\ \beta = e^{i\varphi}\sin\frac{\theta}{2}. \tag{2}$$

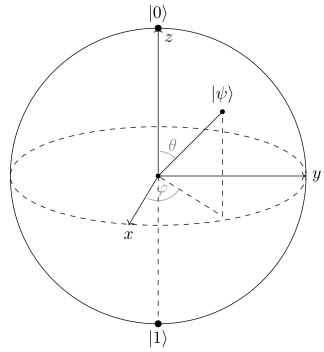


Fig. 1. Bloch-sphere representation of a qubit.

The vector is parametrised by two complex numbers, α and β, which are two probability amplitudes. As probability amplitudes they must satisfy the property $|\alpha|^2 + |\beta|^2 = 1$ [143]. In order to model the value of the qubits in a classical computer, for example for simulation purposes, it is necessary to store both α and β as complex values which requires a higher memory usage compared to storing the non-complex values in classical computers.

2.2 Quantum Gates

Quantum gates are the quantum equivalent of a logical gate. By applying a quantum gate to a qubit or multiple qubits, it is possible to control the probability amplitudes and thus change the state of the qubits.

2.2.1 Single-qubit Gates. Operations on qubits must preserve the norm $|\alpha|^2 + |\beta|^2 = 1$. therefore they are described by 2×2 unitary matrices. Some of the most important single-qubit gates are the Pauli gates:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \; Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \; Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{3}$$

These matrices correspond to the rotation of π radians around respectively the x, y, and z axes of the Bloch sphere.

Another important single-qubit gate is the identity matrix

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \tag{4}$$

which does not affect the value of the qubit,

2.2.2 Multi-qubit Gates. Multi-qubit gates allow n qubits to interact together. The probability amplitudes required to represent an n-qubit system are 2^n. For instance, a 2-qubit system can be represented as:

$$|\psi\rangle = a_{00}\,|00\rangle + a_{01}\,|01\rangle + a_{10}\,|10\rangle + a_{11}\,|11\rangle\,. \tag{5}$$

Therefore, the matrix size for an operation on an n-qubit system is $2^n \times 2^n$. An important class of multi-qubit gates are the controlled gates. The control qubit values determine whether the controlled qubit's or qubits' value(s) will be controlled by the gate. An example of a 2-qubit gate in this class is the controlled-*NOT*, which applies a Pauli X gate on the target qubit if the control

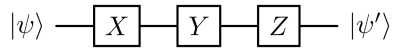


Fig. 2. A single-qubit circuit, which applies the Pauli gates X, Y, and Z in succession.

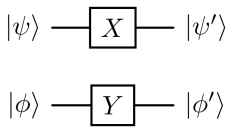


Fig. 3. A two-qubit circuit, which executes a Pauli X gate on the first qubit and a Pauli Y gate on the second qubit.

qubit is $|1\rangle$:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{6}$$

Another example of multi-qubit gate is the Controlled-Z (CZ), which applies a Pauli Z gate on the controlled qubit in the case of a control qubit with state $|1\rangle$.

2.3 Quantum Circuit Execution

Quantum circuits are represented as in the diagram shown in Figure 2. Quantum gates are applied to a qubit in the time domain. Thus, the horizontal line connecting the different gates represent the time line (from left to right) of the different gates applied to the same qubit. In the case of a superconducting computer, this is done by controlling the single qubit with a specific pulse, and the gates are applied in successive instants of time. The circuit depth, as previously mentioned, is the longest path in the circuit, representing the maximum number of gates applied to a qubit. From a mathematical point of view, applying multiple gates can be represented as multiplying the gate matrices with the qubit state vector:

$$|\psi'\rangle = Z \cdot Y \cdot X \cdot |\psi\rangle\,. \tag{7}$$

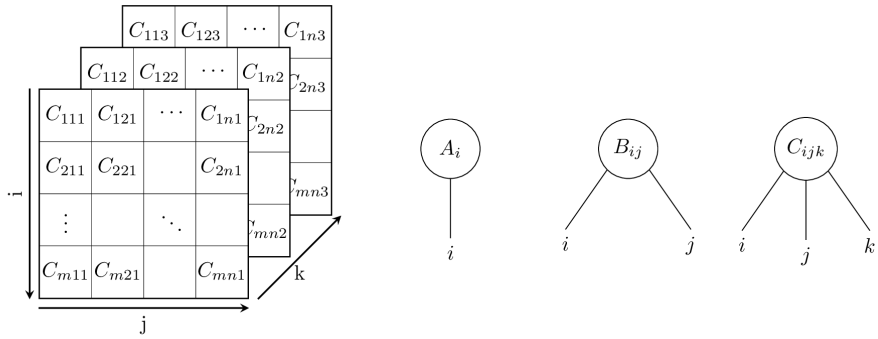
In case of circuits with multiple qubits and gates, the tensor product of two gates is equivalent to executing the gates in parallel. If we consider, for example, the circuit in Figure 3, it is possible to compute the gate matrices:

$$A = X \otimes Y. \tag{8}$$

The complexity of the classical model representation of a quantum circuit scales linearly with the circuit depth and exponentially with the number of qubits. This is because with n qubits there are 2^n possible states, and the size of the resulting matrix representing the parallel gates will be $2^n \times 2^n$. If there are j successive gates, there will be the need for $j \times 2^n \times 2^n$ matrices to represent the quantum circuit.

2.4 Tensor Networks

Some of the simulation methods are based on the representation and execution of the quantum circuit as a tensor network.



(a) (b)

Fig. 4. (a) Example of order-3 tensor, (b) An order-k tensor requires k-indices to be accessed.

For example, a vector A or a matrix B:

$$A = \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{pmatrix} \qquad B = \begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1n} \\ B_{11} & B_{22} & \cdots & B_{2n} \\ \vdots & & \ddots & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mn} \end{pmatrix}, \tag{9}$$

are considered an order-1 tensor and an order-2 tensor, respectively while an order-3 tensor C is represented as shown in Figure 4(a).

A tensor network is a high-dimensional tensor that may be viewed as a graph where each node represents a tensor and the edges represent the connections between them.

A tensor of order-k is an object with k indices and can be represented with k legs [10, 56], as shown in Figure 4(b).

Quantum states can be represented through tensor networks, such as **matrix product states** (**MPS**) [163] or **project entangled pair states** (**PEPS**) [32]. In the simpler case of MPS, a quantum state can be represented by multiplying a tensor with the value of the individual states S_i:

$$|\psi\rangle = C_{S_1 \cdots S_n} |S_1 \cdots S_n\rangle . \tag{10}$$

The tensor C can be decomposed in multiple tensors A_i. We define the size of the tensor A_i with the bond dimension d_i, such that the dimension of the tensors A, i.e., in case of a even number of qubits (n), is $(1 \times 2), (2 \times 2^2), \ldots, (2^{\frac{n}{2}-1} \times 2^{\frac{n}{2}}), (2^{\frac{n}{2}} \times 2^{\frac{n}{2}-1}), \ldots, (2^2 \times 2), (2 \times 1)$. We can fully describe the tensor C such as

$$C_{S_1 \cdots S_n} = \sum_{d_1, \ldots, d_n} (A_1)^{d_1}_{S_1} (A_2)^{d_2}_{S_2 d_1} \cdots (A_n)_{S_n d_{n-1}}. \tag{11}$$

To compute the exact value of the states, we would need $\approx 2^n$ values to describe the MPS, equal to the number of values needed in case of the state-vector simulation. The maximum bond dimension needed to describe the exact value of the states is $2^{\frac{(n-1)}{2}}$ or $2^{\frac{n}{2}}$ depending on whether the number of qubits is odd or even, respectively. By choosing a smaller value for the maximum bond dimension, consequently pruning values in the bigger matrixes, we decrease the complexity of the problem but we introduce an approximation error which is proportional to the entanglement level of the described system [32].

Fig. 5. The tensors D and E are contracted into a single tensor F, with indices i and k.

In order to work with tensors, the main operation is the tensor contraction, shown in Figure 5. Contracting two tensors means performing a summation over an index, which contracts the internal indices between the two tensors, mathematically represented as

$$F_{ik} = \sum_j D_{ij} E_{jk}. \tag{12}$$

The usual approach to handle the tensor networks is to contract the network from order k to a single tensor. When the network contraction is divided in a sequence of binary contractions, the total computational cost is influenced by the choice of the contraction sequence. Working with pairwise contractions typically enables optimal computational performance, as they can be implemented as matrix-matrix multiplication [56].

3 Quantum Computers

Since the beginning of quantum research various algorithms [141] have been explored to verify if a quantum computer could solve some problems faster than classical machines. Examples of such quantum algorithms are Shor's algorithm [166], Grover's search algorithm [74], and quantum simulation [66, 143]. Shor's algorithm, based on the quantum Fourier transform [143], can be used to find the prime factors of an integer. Grover's search algorithm allows to speed up the search for an element which satisfies a certain known property. It allows in the case of a search space of size N to find an element with no prior knowledge about the structure of the information in $O(\sqrt{N})$ operations instead of the $O(N)$ operations required classically [143]. Quantum simulations are simulations of naturally occurring quantum mechanical systems, such as molecules, using quantum computers [143].

Quantum computers utilise different quantum chip technologies, including ion trap [21, 64, 102, 179], neutral atom [17, 85, 195, 201], semiconductor [40], and photonic [108, 131, 145, 160, 172, 210]. This work focuses on the currently most common technology used to build quantum computers: the superconducting architecture [65, 75, 191]. Figure 6 shows the generic structure of a superconducting quantum computer. The high-level scheme for the other quantum chip technologies is similar to the one shown, as there is always the need to control the chip through control signals and read out the output afterwards [160, 179, 195].

3.1 Quantum Chip

The quantum chip, which includes the physical qubits, is the main component of the quantum computer. Depending on the technology, they are controlled and connected in different ways. Connecting multiple qubits together allows for more complex circuits, comprising single- and multi-qubit gates.

As already mentioned in the introduction, one of the main limitations of current physical qubits is their short coherence time. This is the timescale of the exponential decay of the qubit superposition state [4]. The correctness of quantum computers is usually measured in terms of a fidelity for the operations or the output state.

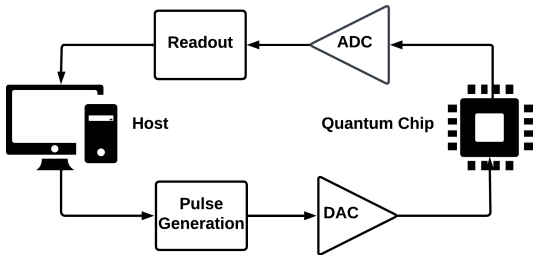


Fig. 6. Basic scheme for a superconducting quantum computer and its control electronics. The control pulses generated are converted from digital to analog through the DAC. The analog output of the quantum chip goes through an ADC and is then read.

Coherence times vary depending on the technology and the "quality" of the qubit. A different metric that allows to compare different technologies and different approaches is the amount of gates that can be executed during the coherence time of a single qubit. In the case of the current superconducting qubit implementations, they have a coherence time ranging from hundreds of microseconds [22, 159, 168], up to close to a millisecond [177]. Consequently, the control system, responsible for providing pulses to activate the qubits and read out their state, must be fast enough to be able to control thousands of gates during this time.

Although some technologies, such as photonics, allow the quantum chips to work at standard room temperature, most of the other approaches require the chips to be cooled down to a temperature close to absolute zero to avoid thermal noise. Therefore, the chips are usually are placed inside cryogenic chambers. This introduces additional challenges related to controlling and communicating with the chips from other components of the quantum computer. If placed inside, the components need to operate at the low temperatures of the chamber and are forced to dissipate minimal power, while if placed outside the chamber, they face a bottleneck in scaling due to the physical limit of input and output cables available. In additional to the physical number of cables, each cable also carries heat into the chamber, thus increasing the thermal noise [112].

3.2 Pulse Generation

The qubit driving mechanism may vary depending on the type of quantum computers we briefly discussed in the previous section. Independently of the chip technology, there is always the need to use control signals, which would be in form of some pulses [160, 179, 195]. In the case of the superconducting architecture, qubits are controlled by radio-frequency signals. Each qubit is characterised by a slightly different resonance frequency, which allows to drive them individually [109, 111, 150]. The specific resonance frequency can vary slightly in time due to interaction with impurities in the qubit environment, affecting the fidelity of operations on each qubit. Therefore, periodic recalibration of the system is necessary [22, 192, 196]. This recalibration process allows to update the correct resonance frequencies used to control the qubit by the pulse generation hardware. Scaling up the number of qubits requires additional inputs to the system, which, as previously mentioned, becomes a major challenge for technologies that require a cryogenic chamber for cooling.

3.3 Readout

After the execution of the quantum circuit, the value of all or some output qubit must be measured. The main challenge is that discrimination between zeros and ones is non-trivial, leading to the development of various readout techniques [8, 27, 46, 104, 111, 138, 156, 174]. The approaches range from improved amplification of the output at the analog level [8, 104, 156] to the introduction of

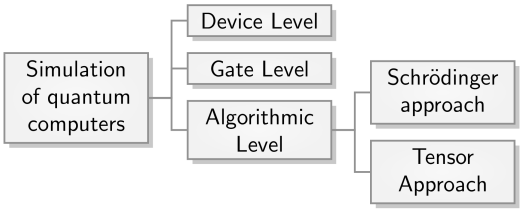


Fig. 7. Taxonomy of simulation of quantum computers.

error correction for fault-tolerant quantum computing using higher-abstraction level techniques such as machine learning [138]. The main issue, as in the case of the pulse generation, is related to managing the connection between the readout system and the quantum chip through the cryogenic chamber, which is one of the major bottlenecks in scaling up the system.

3.4 Compiler

Mapping an algorithm to a quantum circuit is a non-trivial task. To do this, we use specific tools such as a compiler, which must take multiple decisions. Ideally, the compiler would assign each qubit in the algorithm to a physical qubit in the quantum computer, without considering the different error probabilities of each qubit. In this case, all the multi-qubit gate operations would occur between neighbouring physical qubits, giving the compiler freedom to choose any quantum gate operation.

In a real quantum computer, this is not always possible. First, different qubits on the same chip could have different probabilities of error due to the influence of the external environment. When reading the output of a qubit, the output might not always be correct, due to factors such as drifts in calibration, temperature variations, or measurement errors. Therefore mapping the operations to qubits according to their probability of error might increase the fidelity. Additionally, in some architectures, such as superconducting circuits, qubits can only interact with neighbouring qubits, requiring the addition of SWAP operations when direct interaction is not possible [146, 170, 200, 204].

Not every gate might is available on every quantum computer, and the choice of gate can influence the output fidelity. Multiple approaches [29, 37, 39, 120, 126, 149, 171, 190, 197, 204, 211] have been proposed to address this issue, aiming at optimising the mapping of quantum circuit to the available qubits and gates. Thus, it is very important that the compiler considers all the characteristics of the available system as a sub-optimal compilation phase may result in resource under-utilisation, high error rate, and lower fidelity [126].

4 Simulations of Quantum Computers

In this section, we focus on how we can simulate, and optimise the simulation of, the execution of quantum computers. Simulations are becoming an increasingly valuable tool for evaluating and developing various components of the quantum computer stack. As in classical computing, each part of the system requires a distinct approach and level of detail.

Simulations of quantum computers can be classified into the following three main categories, as also shown in Figure 7.

— **Device level** — this is the lowest level of the system and focuses on the materials and the implementation of the qubit. This is comparable to the low-level classical hardware simulation.
— **Gate level** — this is the middle level, where gates are mapped to qubits. This is comparable to the micro-architecture-level classical simulation.

— **Algorithmic level** — this is the highest level of the system, where algorithms are mappped to quantum circuits. This is comparable to the functional-level classical simulation.

Although it is possible to simulate a complete quantum computer for a small number of qubits, this does not scale well for increasing number of qubits. This issue arises at every level and it is better described in each subsection. In general, the simulation execution is both memory- and compute-bound. With scalability in mind, simply trading one for the other is insufficient to achieve the desired performance, as beyond a certain number of qubits, one of the two limitations will become impossible to overcome.

We should notify the reader that in this work, we will not focus on the acceleration of device-level or gate-level simulations, as there are already plenty of solutions [53, 61, 161, 212]. Instead, we will focus on the research field of algorithmic-level simulation.

4.1 Device Level

Device-level simulations help address the mechanical, thermal, and electromagnetic aspects of the problem. In the case of superconducting circuits, some mainstream commercial tools are reported here:

— Ansys Electronic Desktop [90] solves 3D Maxwell equations, and eigen-equations with various boundary conditions, including the option for lumped-element boundary conditions. This is useful for nearly all problems in the electromagnetic domain at RF and microwave frequencies.
— COMSOL Multiphysics [3] is a **Finite Elements Method** (**FEM**) solver of many partial differential equations of physics. It is useful for simulating thermal, mechanical, and electromagnetic problems. London's equations can be solved to simulate the Meissner effect in packages, chips, resonators, as well as effects of kinetic inductance. Heating or cooling of circuits and the impact of mechanical strain on packages or **Printed Circuit Boards** (**PCBs**) can also be simulated.
— SolidWorks [36] can be used for designing of mechanical parts such as nuts and bolts, packages, fixtures, heat sinks, and connectors. Designs can be exported and re-imported in Ansys tools for electromagnetic simulations. It is also possible to export the FEM mesh for use in other FEM-based simulators. FreeCAD is an open-source (python-based) version of this tool.
— InductEX [92] from SunMagnetics Inc. solves London's equations using FEM and is primarily used for extracting inductances in the layout of superconducting circuits, such as **Rapid Single Flux Quantum** (**RSFQ**) logic gates.

Different types of simulators will be used in quantum dot technology, such as Intel Quantum Dot Simulator [100] and nanoHUB Quantum Dot Lab [105, 106]. Although related to different technologies, the simulators can be classified at the same device level. They evaluate the selected material and geometry and calculate information such as 3D visualisation of the confined wave functions, incident light angle and polarisation, and isotropic optical proprieties [105, 106].

4.2 Gate Level

Gate-level simulation focuses on the interaction between a few qubits connected by couplers, which are used to implement quantum gates. Evaluating different configurations is necessary to study and define the actual gates that will be used in quantum circuits.

An example circuit used for gate-level simulation is shown in Figure 8. Simulations at this level are based on solving differential equations derived from the Schrödinger equation [95].

There are different types of simulations at this level. Static gate-level simulation is useful for evaluating the energy levels of qubits and couplers, as well as assessing the interference between

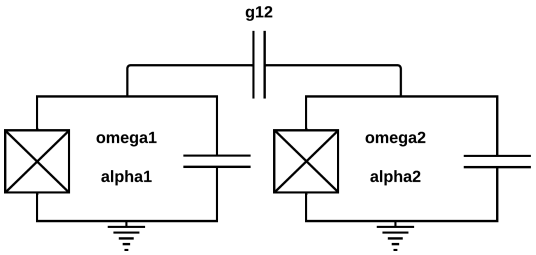


Fig. 8. Simple circuit used for gate-level simulations, composed of two qubits and one coupler to connect them. Each qubit is modelled as a Josephson junction and a capacitor. In the circuit simulations a capacitive coupling is used to connect the two qubits [63, 155].

them. Dynamic gate-level simulation helps evaluate the effects of a pulse used to control the gate. This is crucial for determining the type and frequency of pulse needed on the quantum chip to ensure each qubit responds as expected.

Examples of simulations at this level are CSQR [63, 155] and scQubits [31, 73]. The gate-level simulation acceleration is based on the optimisation of the differential equation solution. Other simulators, such as QuTiP [94, 95], allow for run-time optimisation of the type of solver and hardware configuration, enabling faster simulations or reduced memory usage. Dynamiqs [77] is another simulator that can run on CPU, GPU, and TPU, offering speedup through batching and parallelisation on CPUs and GPUs. Another simulator at this level is QuantumOptics.jl [110], but no information on the acceleration is available.

4.3 Algorithmic Level

Algorithmic-level simulation is useful for testing the correct functioning of potential quantum algorithms. Although we mentioned previously in Section 3 that this work focuses on superconducting architectures, simulation of algorithms is generally platform-independent.

One such algorithm is Shor's 9-qubit error-correction code [166] shown in Figure 9. Unlike device-level simulations, algorithmic-level simulations generally treat the quantum circuit as a sequence of matrix operations [143]. Simulations may include noise, which models external factors such as temperature, electrical interference and additionally device changes in time, that may affect the state of the qubits. Noisy quantum states are represented as a density matrix, which requires more storage than the vector of a pure quantum state. Consequently, including the noise model in the simulation leads to higher accuracy of the simulated system but results in increased problem complexity and thus higher demands on the computational resources [49]. In contrast, we can exploit the fact that a lower simulation accuracy would match the real output of the noisy quantum circuit, therefore leading to lower computational demands [135]. Consequently, noiseless simulations need less memory and compute time, but they are farther from real-world conditions. Nowadays, we are moving towards **Noisy Intermediate-Scale Quantum** (**NISQ**) computing [16, 157]. Introducing noise in the simulations can provide a better correlation between simulation and real-world results. Below is a list of some of the most common algorithmic-level quantum circuit simulation tools available for use:

— Intel Quantum SDK [100]: Provides two different simulators with two different modes of simulation, Generic Qubits and Intel Hardware. The Generic Qubits simulation is a state-vector-based simulation, which uses as a backend the Intel Quantum Simulator, allowing for a qubit-agnostic execution. The Intel Hardware mode uses as its core the Quantum Dot Simulator, which simulates the Intel quantum hardware, currently under development.

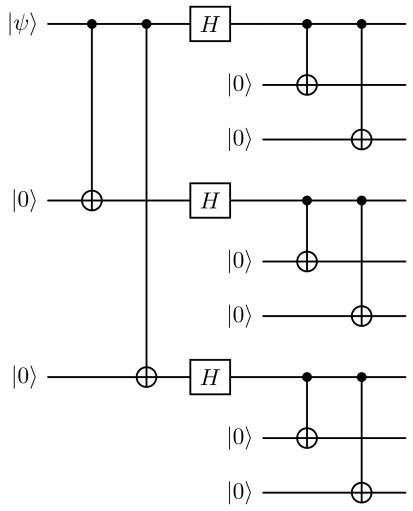


Fig. 9. 9-qubit Shor's code circuit for error correction [167]. Different gates are used in this circuit: multiple Hadamard gates which change the qubit state to the superposition state, and CNOT gates [143].

- QuEST [96]: Simulator using state vectors and density matrices [143] for quantum circuits, which uses multithreading, GPU acceleration, and distribution to optimise the execution on multiple types of devices such as laptops, desktops, and networked supercomputers.
- Qiskit [158]: Open-source SDK for working with quantum computers. Used as the core library for different types of simulations. It allows both for noiseless and exact noisy simulation with Qiskit Aer [158].
- HyQuas [205]: This is a hybrid partitioner-based quantum circuit simulator, optimised for GPUs. It selects the optimal simulation method for different parts of a given quantum circuit. It implements additional solving methods and makes use of distributed simulation.
- qTask [88]: Quantum circuit simulator that focuses on optimising the speed in case of modifications of a small part of the circuit, defined as incremental simulation. When this happens, the state amplitudes (probability amplitudes of the state) are incrementally updated, removing the need for simulating the whole circuit every time.
- PennyLane [14]: Open-source software framework for quantum machine learning, quantum chemistry, and quantum computing. Supports GPU acceleration by making use of NVIDIA cuQuantum SDK [12].
- QuTip-qip [119]: QuTiP quantum information processing package, which offers two approaches to quantum circuit simulation. The algorithmic-level approach is based on the circuit evolution under quantum gates by matrix multiplication, while another approach uses open system solvers in QuTiP [94, 95] to simulate noisy quantum devices.

While gate-level simulations focus on the full time evolution from the start to the end of the gate, algorithmic-level simulations consider only the changes that the gate's corresponding matrix applies to the state vector. Although these simulators provide less detail, they allow for faster simulations or alternatively an increased number of simulated qubits. However, they are also constrained by memory and time. The memory required to simulate a circuit is determined by the number of qubits, the number of gates, the data representation, and the number of times the circuit is executed, which, in the case of a real quantum circuit, corresponds to the number of measurements.

Most simulators are designed for algorithmic-level simulation, which is the furthest from actual physical implementation and allows for evaluating problems from a hardware-agnostic perspective. Quantum simulations at the algorithmic level can be categorised into four main types of approaches:

— Schrödinger-style simulation [143]
— Feynman-style simulation [15]
— Tensor-based simulation [136]
— Stabiliser simulation [2]

While the Schrödinger and Feynman approaches have been the primary methods in the past, there is now a greater focus on exploring the potential of the tensor-based approach.

4.3.1 Schrödinger-style Simulation. Schrödinger-style simulation, also known as state-vector-based simulation, is the mainstream technique for general-case simulation of quantum algorithms, circuits, and physical devices. As described earlier, a quantum state is represented by a vector of complex-valued amplitudes, and the primary approach in this simulation is to store the current state vector and iteratively multiply it by a state transformation matrix [88].

Schrödinger-style simulation resources scale linearly as a function of the circuit depth and exponentially as a function of the number of qubits. Additionally, it allows for relatively straightforward implementations that are commonly used for small and mid-size quantum circuits and device/technology simulations [60].

4.3.2 Feynman-style Simulation. Feynman-style path summation considers each gate connecting two or more qubits in a quantum circuit as a decision point from which the simulation branches. The final quantum state is obtained by summing up the contributions of the results of each branch, which are calculated independently. In comparison to Schrödinger-style simulations, traditional Feynman-style path summation [1] uses very small amounts of memory but doubles the runtime on every (branching) gate. This results in a much longer runtime and does not allow for optimal memory usage, as the number of paths grows exponentially as a function of the decision points. Unlike traditional Schrödinger-style simulation, Feynman's resulting algorithms are depth-limited, making them a good fit for near-term quantum computers that rely on noisy gates [135].

4.3.3 Schrödinger-Feynman Hybrids. A possible approach to leverage both Schrödinger and Feynman approach is Schrödinger-Feynman hybrids [1, 30], proposed in the work of Markov et al. [135]. In the context of nearest-neighbour quantum architectures [7, 18, 103], where the qubit array is partitioned into sub-arrays, the Schrödinger approach is applied to each sub-array. This reduces the memory requirements for k qubits from 2^k to $2^{\frac{k}{2}+1}$, although this introduces a dependency on the number of gates acting across the partition.

The gates are decomposed into a sum of separate terms to allow for independent simulation, but this also results in an increase in execution time. For example, in the case of a CZ gate, the gate can be decomposed into two terms, which results in doubling the run time. However, this increase in run time is still lower compared to the Feynman-style path summation described earlier.

4.3.4 Tensor-based Simulation. The Schrödinger approach typically stores the full state of the qubits, allowing for the simulation of arbitrary circuits. However, the downside is the exponential increase in memory requirements. For an n-qubit system, $O(2^n)$ space is needed.

When using the tensor-based approach, the quantum circuit is described as a tensor network, with each n-qubit gate represented as a rank-$2n$ tensor. This transforms the simulation into a problem of contracting the corresponding tensor network. Tensor network contraction is performed by convolving the tensors until only one vertex remains. For circuits with a large number of qubits and shallow depth (as complexity often grows exponentially with circuit depth), this method is

highly efficient. Thus, using tensor networks allows for the simulation of only one or a small batch of state amplitudes at the end of the circuit. The complexity of the tensor network is constrained by the largest tensor involved in the contraction process.

4.3.5 Stabiliser Simulation. Stabiliser simulations are used for Clifford circuit simulators, quantum circuits that are easy to simulate for classical computers since they only use Clifford gates, start in the computational basis state, and measure in the computational basis. Then Clifford gates can just take you between six points on the Bloch sphere (±1 on the x, y, and z axes). This is described in the Gottesman-Knill theorem [71]. Such circuits are easy to simulate for hundreds, thousands, or more qubits [13, 20, 33, 70, 134]. Qiskit [158] is one of the tools which allows for stabiliser simulations. Stim [68] is also a popular such simulator for studying error correction, made by Google's expert on the topic. IBM also proposed such type of simulation [19]. A famous early work on simulation with stabiliser circuits is presented in [2]. Some works also present a hybrid simulation that combines stabiliser simulation and tensor-network simulation [137]. While the stabiliser simulation is limited to Clifford circuits, several works present ways to modify generic circuits to be able to apply this simulation. This is done by extracting Clifford subcircuits [125, 175] or recompiling the circuit to Clifford gates entirely [68]. The acceleration of stabiliser simulations will not be covered by this survey, as it is limited to Clifford circuits only, while our focus is on accelerating the simulation of generic circuits.

5 Simulation Hardware Platforms

Based on the size of the problem, various hardware platforms with different computational power can be used to run the simulations. Nevertheless, all of them will eventually be limited by either the required memory or the simulation time. The exponential growth of memory and simulation time is the reason for the relevance of the simulation optimisation efforts, which will be discussed later in Sections 6 to 8. A diagram presenting the techniques discussed in this work is depicted in Figure 10.

5.1 Small-scale Quantum Simulation

Currently, an algorithmic-level simulation of 30 qubits with Intel quantum SDK [100] requires up to 34 GB of free RAM. Increasing this by just two additional qubits would require up to 135 GB. This means that with today's technology, even a common personal computer can be used for a moderately small number of qubits. A simulation with another algorithmic-level simulation tool, qTask [88] on a CentOS 64-bit machine with 16 Intel i7 cores at 2.50 GHz and 128 GB RAM, will take on average 991.4 ms to run a 26-qubit Ising model simulation, or 2209.7ms to run a 20-qubit Quantum Fourier transform, with respectively 114.3 GB and 91.2 GB RAM [88]. Personal computers are the most commonly available platform for running simulations, primarily using the CPU and, in some cases, the GPU. Personal computers can also be combined with an external FPGA, which allows off-loading some of the work. As described in Section IV-C, in state-vector simulations, the core operation is matrix multiplication, which, if properly handled due to the use of complex numbers, can be processed inside the FPGA and the results then transferred back to the CPU.

5.2 Medium-scale Quantum Simulation

Scaling is quite an issue in quantum computer simulations, as mentioned earlier. This increase in computational resources can be supported by better simulation hardware, such as workstations. These are typically equipped with state-of-the-art CPUs and one or more GPUs, offering more memory compared to a standard personal computer. This enables the simulation of a larger number of qubits, although still limited due to the exponential scaling of memory requirements. With current

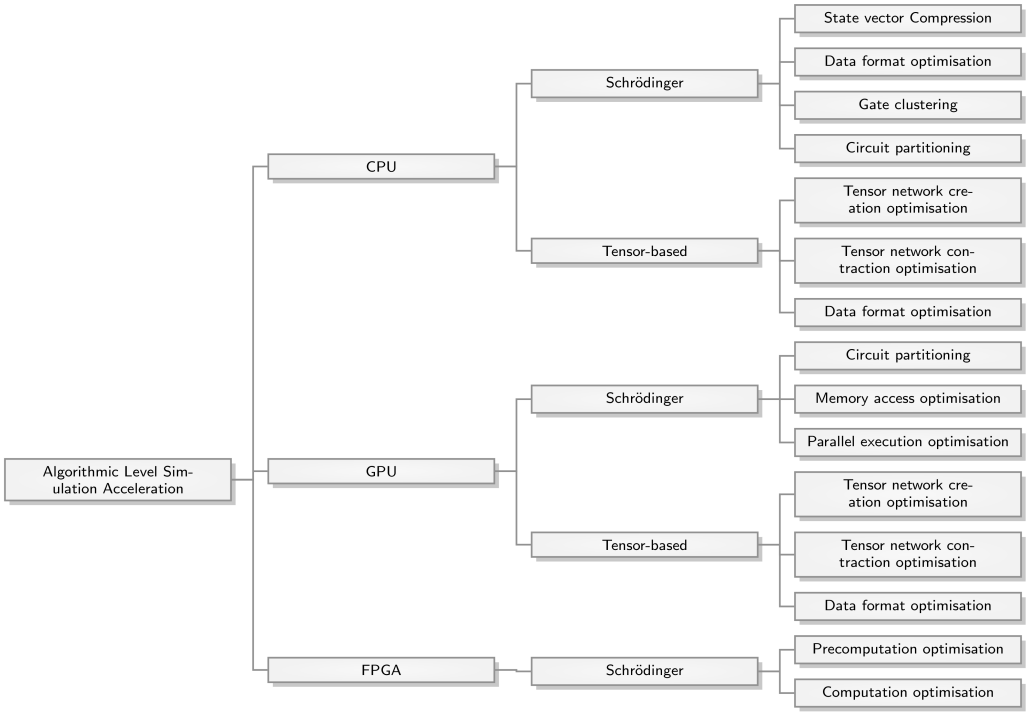


Fig. 10. Summary of acceleration methods.

technology, simulating up to 32 qubits is possible using commercially available workstations [100]. Another factor to consider is the simulation time. Upgrading to workstations, especially those equipped with multiple GPUs, generally enables faster execution if the simulator is optimised for parallel execution. Additional speedup can be achieved if the workstation has a processor with a matrix unit [176], which allows for quicker execution of core operations. The simulation of a quantum Fourier transform with 36 qubits on one node of Argonne [113] requires 192GB of memory and can be run in more than three days [198].

5.3 Simulation Beyond the Medium-scale

If the goal is to run simulations with the highest possible number of qubits or at the fastest speed possible, a high-performance computer is required. An example of this is the execution of 61-qubit quantum circuits [198] on Argonne Theta [113] using qHIPSTER [173], an earlier version of the Intel quantum simulator [100]. The simulation, although optimised, required 0.8 Pb of memory, a level of resources not available on standard laptops or workstations. The advantage of high-performance computers is that they are equipped with multiple nodes, each composed of multiple CPU cores or GPUs, allowing for significantly better performance compared to a single laptop or workstation. The simulation of a Grover search with 61 qubits on 4,096 nodes of Argonne [113] requires 768 TB of memory and can be run in more than 8h [198].

6 Acceleration using CPU

CPU simulation acceleration focuses mainly on scalability, and in particular, most of the work reported here tries to optimise the simulation on big clusters of CPUs, with high-performance computers such as Summit [57], Sierra [114] (Lawrence Livermore National Laboratory), or Theta [113]

Table 1. Summary of the most Important Acceleration Works Performances with CPU

Work	Simulation type	Baseline	Improvement (Scaleup / Speedup)	Benchmark Platform
[130]	Tensor-based	120-qubit QAOA [209]	210-qubit QAOA in 64 s	Argonne Theta Supercomputer
[127]	Tensor-based	81-qubit RQC, 40 depth [26]	100-qubit RQC, 42 depth in 304 s	Sunway Supercomputer
[58]	Schrödinger	30-37 qubits circuits in range range10100 s [76]	up to 5.67× speed-up	Workstation, Frontera Supercomputer
[135]	Schrödinger	45 qubits, 26 depth, 0.5PB mem [81]	45 qubits, 27 depth, 17.4GB mem in 1.4 h	Google Cloud
[60]	Schrödinger	36 qubits in 194.12 s [184]	36 qubits in 94.48 s	Linux Server
[121]	Schrödinger	49 qubits, 27 depth in ≥24 h [154]	49 qubits, 27 depth in 1.49 h	Sunway Supercomputer
[198]	Schrödinger	47 qubits, 2.8PB mem [143]	61-qubit Grover's search, 0.8PB mem	Argonne Theta Supercomputer
[43]	Schrödinger	45 qubits, JUQCS-E [43]	48 qubits in 300 min	K Computer (RIKEN)

For each work, the baseline simulated circuit is shown, as well as, the memory consumed and execution time, whenever reported. The Improvement shows each work's achievements, as presented by the authors, including scaling of the problem, speedup and/or memory reduction.

(Argonne National Laboratory). The optimisations presented are classified by their approach: Schrödinger or Tensor-based. Table 1 summarises all the results.

6.1 Schrödinger-style Simulation Acceleration

As described previously in Section 4.3, accelerating Schrödinger-style simulations consists of speeding up the core matrix multiplication. Different approaches have been taken to achieve improvements.

6.1.1 Baseline Simulation Algorithm. The most mathematically simple way to execute full state-vector simulations for an n-qubits circuit with quantum gates, reported in the work of Fatima and Markov [60], is to

(1) Order the gates left to right
(2) Execute each layer of gates in parallel. All qubits must go through either an actual gate or an identity matrix; therefore pad each gate with an identity matrix of an appropriate dimension via Kronecker products to obtain a $2^n \times 2^n$ matrix
(3) Multiply all matrices in order

This allows the entire circuit to be represented as a matrix, and multiplying it with the input state vector produces the output. Although it is a simple algorithm, it is not the most efficient method in terms of memory utilisation and parallelising computations. Another straightforward approach is to apply each gate directly to the input state vector. Generally, in most works, the simulations are performed in two main steps. The first step is to partition the circuit into multiple sub-circuits, which can be executed in parallel, eventually grouping or reordering gates within the same partition. The second step involves adopting various strategies to optimise the execution. Additional techniques, such as compression and optimising the data format, can further enhance performance.

6.1.2 State-vector Compression. One of the main issues of the Schrödinger simulation, mentioned already before, is memory usage. Storing the full state vectors during the simulation requires an exponentially increasing memory space. A possible solution, used already in other applications that handle vast volumes of data, is the use of data compression techniques.

There are mainly two types of compressions: lossless compression [47, 140, 185] and error-bounded lossy compression [34, 115, 122–124, 162]. Introducing a module in the simulation workflow that compresses the state vectors before storing them in memory and decompresses them when needed helps address the memory usage problem.

Lossless compression is an approach used in works such as Zhao et al. [208]. The authors report an incremental improvement by stacking multiple methods, and compared to the previous method is 13.29% better for the largest number of qubits (34), which can run on their platform. Their work focuses on reducing the data transfer between CPU and GPU, and one of their contribution is the

lossless data compression of non-zero amplitudes, achieved by observing how the state vector after each operation has similar amplitude values. Lossy compression of state vectors is the approach used in works such as Wu et al. [198]. That approach allows to simulate Grover's search algorithm, as well as other quantum algorithms such as **Quantum Approximate Optimisation Algorithm** (**QAOA**) [59] and random circuit proposed by Google [62], with up to 61 qubits. This is achieved by compressing the state vectors and reducing the memory size required from 32 EB to 768 TB. For the data compression, the authors implement an error-bounded lossy compressor, a technique that limits the compression ratio based on the corresponding fidelity loss. It is possible to select the optimal compression strategy during different parts of the simulation, using Zstd [140] alongside an error-bounded lossy compression method developed by the authors [198].

Another approach, presented in [43], is to use an adaptive encoding scheme based on the polar representation $z = re^{i\theta}$ of the complex number z for the state-vector elements. One byte is used to encode the angle $-\pi \leq \theta \leq \pi$ and the remaining bytes encode the value r, after scaling based on the maximum and minimum values of the state vector. This allows for a reduction in the amount of memory required to store the state by a factor of 8, but it requires additional time to perform the encoding and decoding procedure, up to a factor of 3-4.

6.1.3 Data Format Optimisation. While the 64-bit double representation is the most common approach to the data representation [43], there have been efforts to reduce the number of bits required to obtain results with a good enough fidelity. This allows for a lower memory footprint and higher throughput.

The work of Fatima and Markov [60] uses the 32-bit float type representation instead of the more common 64-bit double type, therefore reducing the total amount of necessary resources. The approximation error is controlled by globally keeping track of the changes in amplitude for changes bigger than $\frac{1}{2}$.

This is done because most of the gates do not significantly change the amplitude value. Gates more commonly act on the phase, so the result is not at risk of underflow. An underflow occurs when the result of an arithmetic operation is relatively so small that it cannot be stored in the input operand format without resulting in a rounding error that is larger than usual.

But some gates, such as the Hadamard gates, can change the amplitude by a factor of $\frac{1}{2}$ or $\frac{1}{\sqrt{2}}$, generating underflow issues with the 32-bit float representation. In this case, the change is tracked and included at the end of the quantum circuit, allowing the reduction of memory needed while still obtaining results with good fidelity.

The same research group published a follow-up work [135] where their simulation acceleration techniques are optimised in terms of parallelisation, with a focus on allowing the simulation to be run on generic hardware.

6.1.4 Gate Clustering and Circuit Partitioning. Simulating gates one at a time, as described in the baseline algorithm, is slow because it requires separate memory traversals [60]. A common technique to avoid this issue is to simulate gate clusters in batches. A gate cluster is obtained by combining the matrices representing the gate matrices acting on one or multiple qubits. The common approach is to cluster adjacent gates acting on the same qubit. Google QSim [184] merges each one-qubit gate to some nearby two-qubit gate. Another work [81] applies gate clustering to a larger amount of qubits, up to five, and then multiplies out the obtained gate matrices.

The work of Fatima and Markov [60] creates clusters of q qubits out of the total amount of n qubits that grows as $O(q^2)$. They apply gate clustering by reordering, a technique that loops over the circuit and clusters adjacent gates of the same type or reorders non-adjacent gates to form larger clusters, allowing for optimised algorithms for each type of cluster. The approach of qTask [88] is

to partition a state vector into a set of blocks, with each partition spawning one or multiple tasks performing gate operations on designated memory regions. This, as the previous techniques, allows to enable inter-gate operation parallelism due to the breaking down of the gate dependencies.

Another strategy presented in [121] is the technique named implicit decomposition. The target circuit is efficiently partitioned into different parts with a focus on balancing the memory requirements for each one of them. This efficient partitioning allows to save memory space compared to storing the entire state vector, because it is not necessary to have all the amplitudes after the individual calculations. They additionally propose a dynamic algorithm to select the optimal partition scheme.

6.1.5 Memory Access Optimisation. Given that the execution is dominated by matrix multiplication operations, an obvious optimisation is to do them in a cache-friendly way, resulting in improved performance [58]. Memory locality can be exploited with gate clustering. If paired-up gates act on qubits as closely as possible, it reduces memory strides (distance between two successive elements of an array in memory) when simulating gate pairs acting on less significant bits [60].

Another technique, described in [60], is cache blocking, where rather than applying pairs of qubit gates in separate passes, such pairs, acting on different qubits, are reordered and applied partially in different orders. Each state vector is divided into chunks, which fit in the L2 cache, and for each chunk, multiple non-overlapping pairs of one-qubit gates and an occasional unpaired gate are applied to it. This reduces the cache misses and improves the performance. The work of Fang et al. [58] tackles the memory locality with their hierarchical simulation framework. The authors consider that scaling up means the working set size of the simulation set would exceed the cache size of a modern CPU. To address this, the input circuit is executed as a sequence of sub-circuits, each containing a portion of the original gates, allowing for better locality.

6.1.6 Parallel Execution. Efficiently handling the parallel execution of different gates or tasks is crucial to capitalising on the advantages introduced by the circuit modifications mentioned earlier. Most works utilise a multi-CPU approach with OpenMP and MPI-based CPU simulations. The work of Fatima and Markov [60] explains in detail how to exploit CPU architecture to simulate different clusters, exploring data-level parallelism.

6.2 Tensor-based Simulation Acceleration

Tensor network-based simulations are divided into two main steps. The first one is to represent the quantum circuit as a tensor network. The second step is to apply the contraction to the tensors, as described previously in Section 2.4. We now present the different acceleration solutions for each step of the simulation.

6.2.1 Tensor Network Creation. For the first step of the simulation, different approaches can be taken depending on the type of algorithm and the structure of the quantum circuit. In the case of **Random Quantum Circuits** (**RQC**), a viable solution is using PEPS representation of quantum states from many-body quantum physics [127]. This representation, according to [127] works for $2N$ by $2N$ lattice types, but to work with different structures, the generation of the best contraction path (choice of which nodes to contract at different steps of the simulation) becomes a bigger issue. A possible approach is to use the CoTenGra software [72] to look for the best path. It uses a loss function that combines the considerations for both the computational complexity and the compute density, which are determinant factors for the performance on a many-core processor [127]. Another approach is to use ordering algorithms. There are several available ordering algorithms, classified as

— Greedy algorithm: usually used as a baseline, contracts the lowest-degree vertex in a graph.
— Randomised greedy algorithms: The computational cost of the contraction at each step is function of the maximum number of neighbours for a node. Minimising this by choosing the

optimal contraction order allows for a decreased computation cost. This approach can do so without prolonging the run time [130].
— Heuristic solvers: Attempt to use some global information in the ordering problems. Example of this are QuickBB [69] and Tamaki's heuristic solver [181].

6.2.2 Tensor-network Contraction. Depending on the optimisations done during the tensor creation step, the contraction can be more or less efficient. During the contraction phase, an optimal slicing method is required to divide the tensor network into different clusters. This allows an efficient parallelisation of the computations, to efficiently process all the sub-tasks of the circuit across all the available nodes. This is necessary to balance the compute and storage costs. In the graph representation, the contraction of the full expression is done by consecutive elimination of graph vertices. The consequence is that the vertex is removed from the graph and the neighbours are connected [130]. The slicing of a tensor over an index means evaluating many variables while keeping one of them constant. Finding the optimal index to slice is the focus of this process. An approach to this is to select the best index after each step of the slicing, a method known as step-dependent slicing [130].

Another approach is to divide the contraction into two steps, the index permutation of the tensors as a preparatory step and the second step is the following matrix multiplication to achieve the contraction results [127]. Permutation of indices is generally required to convert the tensor contractions into efficient matrix multiplications. In the case of tensor contraction on many-core processors with a high compute density of high-rank tensors, it is important to reduce the permutation cost to reduce the movement of data items with strides in between, which is unfriendly for current memory systems [127, 188]. A proposed approach is to use fused permutation and multiplication, which means using different compute processing elements in a collaborative way [127], where each CPU reads its corresponding data block in a regular, constant stride, pattern, thus achieving a high utilisation of the memory bandwidth. Parallelisation of the tensor computation is done after the slicing. A possible approach is to use a two-level parallelisation architecture. On a multi-node level, the partially contracted full expression is sliced over n indices and distributed to 2^n message-passing interface ranks. Node-level parallelism over CPU cores is done using system threads. For every tensor multiplication and summation, the input and output tensors are sliced over t indices. The contraction is then performed by 2^t threads writing results to a shared result tensor [130].

6.2.3 Data Format Optimisation. To achieve a high accuracy in the computation of the simulated output state compared to the actual output state it is necessary to represent the data with a correct amount of bits. The default approach is to use 64-bit double (floating-point) representation [43]. In the case of RQCs, the work of Yong et al. [127] proposes an adaptive precision scaling, which adjusts the data precision to single or half-precision dynamically depending on the degree of the sensitivity of different parts of the computation.

6.2.4 Maximum Bond Dimensions Truncation Optimisation. As described in Section 2.4, choosing a maximum bond dimension lower than the one required to compute exact results, allows a lower complexity of the simulation. The optimised maximum bond dimension is the minimum size which does not introduce an approximation error higher than the one allowed by the selected quantum circuit. The optimal maximum bond dimension is complicated to calculate, as the maximum accepted error depends heavily on the quantum algorithm being simulated, and the relationship between the maximum bond dimension and the approximation error is a function of the entanglement of the simulated circuit and of the algorithm used to generate the tensor representation [32]. Algorithms such as DMRG [163, 193] are one of the possible candidates for this optimisation.

Table 2. Summary of the most Important Acceleration Works Performances with GPU

Work	Simulation type	Baseline	Improvement (Speedup)	Benchmark Platform
[12]	Multiple	53-qubit RQC, depth 10 [148], GPU	4.05× on average	NVIDIA H100 GPU, NVIDIA A100 GPU
[12]	Multiple	53-qubit RQC, depth 10 [144], GPU	4× on average	NVIDIA H100 GPU, NVIDIA A100 GPU
[12]	Multiple	53-qubit RQC, depth 10 [82], CPU	547.35× on average	NVIDIA H100 GPU, NVIDIA A100 GPU
[194]	Tensor based	42 qubits in 1965.52 s [43], CPU	10.0×	NVIDIA A100 GPU
[129]	Tensor based	30 qubits, depth 4 in 246 *s* [82], CPU	175.7×	GPU
[48]	Schrödinger	35-qubit QFT [50], GPU	5×	6× NVIDIA V100 GPU
[118]	Schrödinger	26 qubits in less than 10ms [96, 178]	$\geq$ 10×	GPU Cluster
[49]	Schrödinger	22-qubit QFT [158]	up to 10×	GPU

For each work, the speedup reported by the authors is presented in the Improvement column.

6.3 Other Acceleration Approaches

Alternatively to what was described previously, some works exploit other approaches for optimisations, such as QPUs [182, 194] and different memory technologies. For example, CutQC [182] employ classical CPUs interacting with a small quantum computer. This hybrid computing approach enables the evaluation of larger quantum circuits that cannot be executed on a classical computer alone. It offers better fidelity and allows the simulation of larger circuits than either approach could achieve independently. SnuQS [147] focuses on the full utilisation of the storage device connected to the system. That work aims at achieving maximum I/O bandwidth by employing memory management and optimisation techniques, resulting in better performance compared to DDR4 DRAM main-memory-only systems. Other approaches that make use of multiple hardware platforms can be found in the literature [41, 50, 52, 54, 55, 117, 188, 203]

7 Acceleration using GPU

Accelerating quantum computer simulations using GPUs by offloading workloads partially or fully from the CPU to GPU has been an effective approach for many years. Early works, such as the one by Gutierrez et al. [79], attempted this over 10 years ago. Similar to CPU simulations, GPU simulations can also be classified according to the classification defined in Section 4. The results from the most relevant works are presented in Table 2.

7.1 Schrödinger-style Simulation Acceleration

As seen in Section 6, the core operations of quantum circuit simulation are small matrix multiplications. If the data dependencies between these operations are limited or nonexistent, and they can be parallelised, GPUs can perform them with much higher throughput compared to CPUs. It is still important, even for GPU execution, to focus on data locality to minimise memory access misses [205]. In GPU simulation acceleration, each work focuses on one or both of two steps: first, partitioning the circuit into sub-circuits or tasks; second, accelerating the computation.

7.1.1 Circuit Partitioning. As seen for the CPU in Section 6, also in the GPU approach the first step followed is to partition the circuit in order to enable a more efficient parallel execution. HyQuas [205] proposes a circuit-aware partition strategy and a high-accuracy performance model that guides the partitioning. This allows to obtain a near-optimal partition of a given quantum circuit into different groups and select an optimal method to compute each one. CuStateVec, a library of the cuQuantum SDK [12], which focuses on state-vector simulation acceleration for GPU, uses gate fusion [173]. Numerous small gate matrices are fused into a single multi-qubit gate matrix, which can then be computed in one shot instead of performing multiple computations. This allows for improved performance in cases where both the high compute performance and high memory bandwidth of the GPU are used.

7.1.2 Memory Access Optimisation. Different approaches can be used to address the data locality issue. ShareMem [78] method considers a circuit partitioned in several gate groups, which are applied only to a subset of k qubits out of the n total qubits. The total 2^n state vector values can be split into fragments of size 2^k, which can be stored in GPU shared memory, and each fragment can be mapped to one GPU thread block. The thread block can load the fragment from the global memory to the shared memory, apply the gates on it, and store back the fragment in the global memory. This method performs better than the following BatchMV approach in the case of a sparse part of the circuit [205].

The BatchMV [180] method is based on the idea that if all the target and control qubits of a gate group come from the same k qubits, it is possible to merge these gates into a k-qubit gate. This allows to divide the simulation into 2^{n-k} matrix-vector multiplication tasks, each one with a $2^k \times 2^k$ gate matrix to a state vector of 2^k values. The index of the values only differs on these k positions, improving the data locality. This method performs better than the previous in the dense part of the circuit. HyQuas [205] automatically selects the best approach depending on the part of the circuit between OShareMem, an improved version of ShareMem [78] and TransMM, which performs better than BatchMV in their framework. TransMM transposes the quantum state, allowing to treat the gate-applying operation into a standard **GEneral Matrix Multiplication** (**GEMM**) operation, which can be accelerated by highly optimised libraries such as cuBLAS and Tensor Cores.

7.1.3 Parallel Execution. As seen in Section 6, the main way to approach distributed computing is with OpenMP or MPI. It is important to maximise GPU usage while minimising data exchange between the CPU and GPU, as frequent amplitude exchanges between them introduce significant data movement and synchronisation overhead [208]. Qubit reordering [44] is also a technique used by cuStateVec [12] to address the challenge of applying a gate onto a global indexed qubit. The approach involves moving the qubit from a global index to a local index, allowing a single GPU to compute the state vector. On a more broad approach, there are even works proposing a novel multi-GPU programming methodology, such as the work from Li et al. [118], which constructs a virtual BSP machine on top of modern multi-GPU platforms.

7.2 Tensor-based Simulation Acceleration

As reported in the work of Vincent et al. [189] a major issue in quantum computer simulations is the under-utilisation of GPUs and CPUs in supercomputers. They further predict an increase in inefficiencies due to the increase of parallelism and heterogeneity in exascale computers, with billions of threads running concurrently [83, 107]. Future supercomputers will see an increase in on-node concurrency rather than the number of nodes, with large multi-core CPUs and multiple GPUs per node [51].

7.2.1 Tensor Network Creation. For this step of the simulation on GPUs, some works [189] also use CoTenGra [72], as already mentioned in Section 6, to compute high-quality paths and slices for tensor networks. Additionally, there is the possibility of reusing common calculations, as described in Ref. [189], which avoids adding duplicate tasks during the slicing, therefore leading to a task graph with shared nodes.

7.2.2 Tensor Network Contraction. A possible approach, presented for Jet [189] is to use the transpose-transpose-matrix-multiply method. The basic building block of the task-dependency graph is the pairwise contraction of two tensors. This can be decomposed into two independent (partial) tensor transposes and a single matrix multiplication [128]. While for the CPU they use the qFlex [7, 187, 188] transpose method, for the GPU they use cuTENSOR [12], used also in the work of Shah et al. [164].

Another approach to tensor contraction is the bucket contraction algorithm by Dechter [45], as described in QTensor [129] and used also by Shah et al. [164], where an ordered list of tensor buckets (collection of tensors) is created to contract the tensor network. Each bucket corresponds to a tensor index, the bucket index. Buckets are then contracted one by one. The contraction of a bucket is performed by summing over the bucket index, and the resulting tensor is then appended to the appropriate bucket. The number of unique indices in aggregate indices of all bucket tensors is called a bucket width. Memory and computational resources of a bucket contraction scale exponentially with the associated bucket width. It is possible to improve this method by ordering the buckets first and then finding the indices that can be merged before performing the contraction. This allows for a smaller output size and larger arithmetic intensity as presented by Lykov et al. [129]. An issue with the bucket elimination algorithm is that tensors can grow too large to fit in memory, so a solution proposed by Shah et al. [164] is to introduce data compression. Their work proposes a GPU-based lossy compression framework that can compress floating-point data stored in quantum circuit tensors with optimised speed while keeping the simulation result within a reasonable error bound after decompression. The compressed data can be decompressed when the tensors are needed during the computation.

An additional approach is to use a partitioning method, which is optimal for each part of the circuit, depending on if it is sparse or dense, as mentioned already in the Schrödinger approach in the same work of Zhang et al. [205]. OShareMem method [205], an optimised version of the ShareMem method [78], deletes redundant computation, reduces data indexing overhead, and uses a new layout to access the shared memory faster. Another approach is the TransMM method [205], which converts a set of special matrix-vector multiplications in quantum circuit simulation into GEMM to take advantage of highly optimised GEMM libraries and hardware-level GEMM compute units like Tensor Cores. The work of Zhang et al. [205], as mentioned before, introduces an automatic selection method for the approach to be used, by using OShareMem for sparse parts of the circuit, and TransMM method for the dense parts of the circuit, allowing for further speedup.

7.2.3 Data Format Optimisation. As already mentioned for the CPU optimisations, using smaller precision for the data and calculations leads to improved performance. Although the works that try to accelerate the tensor-based acceleration on GPU do not try to propose any optimisation regarding the data format, they pick slightly different configurations. Most of the works use double (64-bit) floating-point precision [164, 194, 205], while other works, such as Jet [189], use different precision such as single (32-bit) floating-point precision.

7.3 Other Acceleration Approaches

Some simulation tools allow for the acceleration of quantum circuit simulation on GPU by focusing more on the optimal usage of the available hardware. An example of this is cuQuantum [12], which provides composable primitives for GPU-accelerated quantum circuit simulation, including distributed computing on multiple GPUs. TensorLy-Quantum [152] is a quantum library with direct support for tensor decomposition, regression, and algebra. It additionally provides built-in support for Multi-Basis Encoding for MaxCut problems [151] and was used to develop Markov chain Monte Carlo-based variational quantum algorithms [153].

8 Acceleration using FPGA

Although FPGAs are more limited in terms of resources and frequency when compared to CPUs and GPUs [35], they allow for more flexibility, leading to architectures better fitted to the problem [206]. While less scalable compared to the implementations with the previous hardware platforms, FPGAs

excel when working with very limited resources, such as simulations running on an off-the-shelf personal computer.

It is worth noting that some of the reported work, especially early efforts, focused on emulating quantum hardware rather than serving as an acceleration platform for quantum computer simulation, where only a few core operations are offloaded to the FPGA.

In the former case, the FPGA acts as a quantum computer itself, with which the host system must interface. Some example of this emulation approach include the early works of Khalid et al. [101] and Aminian et al. [6]. This same approach has been followed more recently in terms of work of Zhang et al. [207]. These approaches are interesting but can only be applied for problems requiring only a very limited number of qubits.

More recent works focus on running only the most compute-heavy operations on the FPGA. FPGA-accelerated simulation mainly focuses on the previously discussed Schrödinger approach. The two main steps of the acceleration are the pre-computation of the circuit matrix and the matrix computation.

8.1 Pre-computation Optimisation

In the pre-computation step, various optimisations have been proposed. The work of Jungjarassub and Piromsopa [99] introduces an algorithm that does not directly multiply all the gates, but first checks for special cases. Depending on the combination of gate types and the transformation they apply, they can avoid executing certain multiplications in certain specific cases.

A similar approach is used in the work of Hong et al. [87], which checks for the matrix values before multiplication, and in the case where ones or zeros are present, avoids processing through the actual multiplication module. This work focuses on the computation of a 2×2 gate regardless of the circuit. It first groups multiple gates into a single one when possible, then simulates the resulting gates one by one. Although this approach does not prioritise parallel execution, it allows for the execution of larger circuits on hardware with limited resources.

8.2 Computation Optimisation

Different approaches have been proposed to speed up the actual calculation. In the case of gates that generate phase rotation, it is possible to use a look-up table to collect sine and cosine values instead of calculating them directly [99]. Look-up tables are also evaluated in the work of Mahmud and El-Araby [132], which replaces any complex calculations with a simpler array-indexed operation. This is further optimised by storing only the value relative to the actual input vectors which are necessary for the simulation.

Another optimisation to save memory operations is to check whether the state vector changes after a gate is applied before saving it back to memory. If no changes are detected, the memory-write can be avoided as is done in the work of Hong et al. [87]. The work of Mahmud and El-Araby [132] evaluates different hardware architectures to identify the best-performing one for the **Complex Multiply and ACcumulate** (**CMAC**) operation, which is the core of simulation. They evaluate a single CMAC unit which processes all the operations (optimised for area but with low throughput), N-concurrent-CMAC units (optimised for throughput but requires a higher number of CMAC instances) and a dual-sequential-CMAC architecture (two CMAC instances connected sequentially, with computation and data write operations overlapped). Lastly, they also propose a kernel-based emulation model useful in case of a repeated set of core operations. The follow-up work by Mahmud et al. [133] proposes an additional approach that improves the scalability of the emulation. Instead of using a look-up table, which sacrifices area for speed, or using dynamic generation (generation of the algorithm matrix elements at compile-time), which sacrifices speed for area, they propose a stream-based CMAC. They stream the algorithm matrix elements at run-time, meaning the

Table 3. Hardware-aware Optimisation Techniques

Technique	Improves	Hardware support
Mixed-precision operations	compute and data storage	CPU [80], GPU [86]
Arbitrary-precision operations	compute and data storage	FPGA [42]
Lossy data compression	data storage	CPU (software), GPU (software), FPGA (compression engine) [28]
Thread-/Task-level parallelism	compute	CPU, GPU
Vector (SIMD) instructions	compute	CPU with vector support [116]
Matrix operations	compute	CPU with MMU [142], GPU tensor cores [89], FPGA Systolic Array [165], AI engines [98]

operation's cost is typically the I/O channel latency between the control processor and the FPGA, which is negligible compared to the time required for processing the algorithm matrix.

9 Summary and Future Directions

In this work, we examined different approaches to speeding up and reducing the memory usage for the simulation of quantum computers. The most important results, reported in Table 1 for the CPU-related works and in Table 2 for the GPU-related works, show how this is possible with multiple different approaches.

Focusing on the hardware-aware optimisations for the simulation problem and challenges, we present in Table 3 a summary of the different techniques, what they aim at improving, and which hardware support is best for that purpose. For each hardware support we also include a reference to a sample work showing a representative implementation of the technique in domains other than quantum computing.

The first group of optimisations aim at improving both the computation and data storage by changing the data and operations to use reduced precision. This has obvious benefits, but the challenge is to control the error introduced, since it is an optimisation that loses information from the original 64-bit double floating-point precision. For these optimisations, FPGAs can offer the benefit to implement hardware that can operate on data of arbitrary precision (not necessarily conforming to standard sizes). Using reduced-precision data and operations is an optimisation that is successfully being exploited in the **Machine Learning** (**ML**) domain currently by applying quantisation of the data values [67].

The second group of optimisations are those focusing on reducing the data storage and are based on data compression techniques. In this case, since the goal is to reduce the data size considerably, lossless techniques are not enough and thus a lossy technique needs to be explored. This technique again comes with the price of reduced accuracy since there is a reduction of information after compressing the data. Several techniques have been applied in the past for scientific applications [23] and in order for the techniques to be applicable with reduced latency, it is important to have a hardware compression module, implemented via a dedicated FPGA [28, 186] or GPU kernel [208].

The third group of optimisations focus on lowering the execution time by leveraging parallel processing and/or using hybrid configurations with dedicated hardware units for some demanding operations. Thread and task parallelism is an effective technique to reduce the execution time by exploiting the existing parallel hardware resources in CPUs and GPUs. **Single-Instruction-Multiple-Data** (**SIMD**) instructions are used to exploit parallelism at a finer-grained scale, at the instruction level. When using SIMD instructions we are using multiple computational units to perform multiple operations in the same cycle. SIMD support is now common in most commodity processors, but the other operations that are very relevant to these simulations are matrix operations, which usually require dedicated matrix units for acceleration. Since matrix operations are also very relevant for AI, there has been a recent increase in products providing hardware support for these operations [169]. Namely, some CPUs and GPUs now have dedicated matrix units and/or neural

acceleration units. Also, the Google TPU [98] used for both ML training and inference is basically a hardware accelerator for the matrix-matrix operations. The use of FPGAs in this case is also very relevant as they can be used to implement matrix-matrix operations for certain non-standard matrix dimensions. Also, as mentioned previously, FPGAs could be exploited to directly implement matrix-matrix operations for complex numbers [132].

Considering all of the above, we believe that in the future there is a need to invest in the development of more effective data compression techniques, a more flexible use of data precision, and hybrid approaches. Since the market share for the simulation of quantum computers is much smaller than for AI applications, we do not expect an explosion of accelerators as has been observed in the recent years for the ML domain. As such, it is unlikely that we will see many hardware accelerators dedicated to the simulation of quantum computers. But the developments in the ML domain can be leveraged to deliver benefits to this domain too, since the critical operations are basically the same: matrix-matrix operations. As such, we expect in the future to see the use of ML accelerators for improving the performance of quantum computer simulations. Lastly, it is very interesting to already see some development using FPGAs and we expect the trend will be to see more and more FPGA-based dedicated hardware to solve specific operations on specific data types in a very efficient way.

10 Conclusions

Quantum computer simulators play an extremely important role in helping the development of new algorithms and hardware for the promising quantum computing paradigm. While several approaches and simulators have been proposed, the characteristics of the problem make it extremely hard to scale to systems with larger number of resources, required for solving more complex problems. The main challenges of quantum computer simulations are the large amount of data that must be stored in memory, with sufficient precision to produce results with acceptable accuracy, and the execution time, which grows exponentially with the increased number of qubits. In this work, we presented a review of existing tools and approaches for systems with CPUs, GPUs, and FPGAs, with a focus on how hardware-aware optimisations can help address the challenges. CPU-based works mainly focused on reducing the memory requirements, thus increasing the number of simulated qubits, while the GPU-based ones focused mainly on the speed-up. Based on this study, we showed the future directions for hardware-aware optimisations, including the use of accelerators designed for other domains but addressing similar problems and the development of FPGA-based hardware accelerators for quantum computer simulation.

References

[1] Scott Aaronson and Lijie Chen. 2017. Complexity-theoretic foundations of quantum supremacy experiments. In *Proceedings of the 32nd Computational Complexity Conference.* Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, DEU, 67 pages.

[2] Scott Aaronson and Daniel Gottesman. 2004. Improved simulation of stabilizer circuits. *Physical Review A–Atomic, Molecular, and Optical Physics* 70, 5 (2004), 052328.

[3] COMSOL AB. 2024. COMSOL Multiphysics. (2024). Retrieved from https://www.comsol.com/

[4] Tahereh Abad, Jorge Fernández-Pendás, Anton Frisk Kockum, and Göran Johansson. 2022. Universal fidelity reduction of quantum operations from weak dissipation. *Physical Review Letters* 129, 15 (2022), 150504. DOI: https://doi.org/10.1103/PhysRevLett.129.150504

[5] Google Quantum AI and Collaborators. 2025. Quantum error correction below the surface code threshold. *Nature* 638, 8052 (2025), 920–926.

[6] Mahdi Aminian, Mehdi Saeedi, Morteza Saheb Zamani, and Mehdi Sedighi. 2008. FPGA-based circuit model emulation of quantum algorithms. In *Proceedings of the 2008 IEEE Computer Society Annual Symposium on VLSI.* IEEE, 399–404.

[7] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (2019), 505–510. https://doi.org/10.1038/s41586-019-1666-5

[8] Jose Aumentado. 2020. Superconducting Parametric Amplifiers: The State of the Art in Josephson Parametric Amplifiers. *IEEE Microwave Magazine* 21, 8 (2020), 45–59. DOI: https://doi.org/10.1109/MMM.2020.2993476

[9] Medina Bandic, Sebastian Feld, and Carmen G. Almudever. 2022. Full-stack quantum computing systems in the NISQ era: Algorithm-driven and hardware-aware compilation techniques. In *Proceedings of the 2022 Design, Automation and Test in Europe Conference and Exhibition*. IEEE, 1–6.

[10] Mari Carmen Bañuls. 2023. Tensor network algorithms: A route map. *Annual Review of Condensed Matter Physics* 14, 1 (2023), 173–191.

[11] Bela Bauer, Sergey Bravyi, Mario Motta, and Garnet Kin-Lic Chan. 2020. Quantum algorithms for quantum chemistry and quantum materials science. *Chemical Reviews* 120, 22 (2020), 12685–12717. DOI: https://doi.org/10.1021/acs.chemrev.9b00829 arXiv:https://doi.org/10.1021/acs.chemrev.9b00829

[12] Harun Bayraktar, Ali Charara, David Clark, Saul Cohen, Timothy Costa, Yao-Lung L Fang, Yang Gao, Jack Guan, John Gunnels, Azzam Haidar, et al. 2023. cuQuantum SDK: A high-performance library for accelerating quantum science. In *Proceedings of the 2023 IEEE International Conference on Quantum Computing and Engineering*. IEEE, 1050–1061.

[13] Ryan S. Bennink, Erik M. Ferragut, Travis S. Humble, Jason A. Laska, James J. Nutaro, Mark G. Pleszkoch, and Raphael C. Pooser. 2017. Unbiased simulation of near-Clifford quantum circuits. *Physical Review A* 95, 6 (2017), 062337.

[14] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Shahnawaz Ahmed, Vishnu Ajith, M. Sohaib Alam, Guillermo Alonso-Linaje, B AkashNarayanan, Ali Asadi, et al. 2018. Pennylane: Automatic differentiation of hybrid quantum-classical computations. (2018). arXiv:1811.04968. Retrieved from https://arxiv.org/abs/1811.04968

[15] Ethan Bernstein and Umesh Vazirani. 1997. Quantum complexity theory. *SIAM J. Comput.* 26, 5 (1997), 1411–1473. DOI: https://doi.org/10.1137/S0097539796300921 arXiv:https://doi.org/10.1137/S0097539796300921

[16] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S. Kottmann, Tim Menke, et al. 2022. Noisy intermediate-scale quantum algorithms. *Reviews of Modern Physics* 94, 1 (2022), 015004. DOI: https://doi.org/10.1103/RevModPhys.94.015004

[17] Dolev Bluvstein, Simon J. Evered, Alexandra A. Geim, Sophie H. Li, Hengyun Zhou, Tom Manovitz, Sepehr Ebadi, Madelyn Cain, Marcin Kalinowski, Dominik Hangleiter, et al. 2024. Logical quantum processor based on reconfigurable atom arrays. *Nature* 626, 7997 (2024), 58–65. https://doi.org/10.1038/s41586-023-06927-3

[18] Sergio Boixo, Sergei V. Isakov, Vadim N. Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J. Bremner, John M. Martinis, and Hartmut Neven. 2018. Characterizing quantum supremacy in near-term devices. *Nature Physics* 14, 6 (2018), 595–600. DOI: https://doi.org/10.1038/s41567-018-0124-x

[19] Sergey Bravyi, Dan Browne, Padraic Calpin, Earl Campbell, David Gosset, and Mark Howard. 2019. Simulation of quantum circuits by low-rank stabilizer decompositions. *Quantum* 3 (2019), 181.

[20] Sergey Bravyi and David Gosset. 2016. Improved classical simulation of quantum circuits dominated by Clifford gates. *Physical review letters* 116, 25 (2016), 250501.

[21] Colin D. Bruzewicz, John Chiaverini, Robert McConnell, and Jeremy M. Sage. 2019. Trapped-ion quantum computing: Progress and challenges. *Applied Physics Reviews* 6, 2 (2019).

[22] Jonathan J. Burnett, Andreas Bengtsson, Marco Scigliuzzo, David Niepce, Marina Kudra, Per Delsing, and Jonas Bylander. 2019. Decoherence benchmarking of superconducting qubits. *npj Quantum Information* 5, 1 (2019), 54. DOI: https://doi.org/10.1038/s41534-019-0168-5

[23] Franck Cappello, Sheng Di, Sihuan Li, Xin Liang, Ali Murat Gok, Dingwen Tao, Chun Hong Yoon, Xin-Chuan Wu, Yuri Alexeev, and Frederic T. Chong. 2019. Use cases of lossy compression for floating-point data in scientific data sets. *The International Journal of High Performance Computing Applications* 33, 6 (2019), 1201–1220.

[24] M. Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, et al. 2021. Variational quantum algorithms. *Nature Reviews Physics* 3, 9 (2021), 625–644. DOI: https://doi.org/10.1038/s42254-021-00348-9

[25] M. Cerezo, Guillaume Verdon, Hsin-Yuan Huang, Lukasz Cincio, and Patrick J. Coles. 2022. Challenges and opportunities in quantum machine learning. *Nature Computational Science* 2, 9 (2022), 567–576. DOI: https://doi.org/10.1038/s43588-022-00311-3

[26] Jianxin Chen, Fang Zhang, Cupjin Huang, Michael Newman, and Yaoyun Shi. 2018. Classical simulation of intermediate-size quantum circuits. (2018). arXiv:1805.01450. Retrieved from https://arxiv.org/abs/1805.01450

[27] Liangyu Chen, Hang-Xi Li, Yong Lu, Christopher W. Warren, Christian J. Križan, Sandoko Kosen, Marcus Rommel, Shahnawaz Ahmed, Amr Osman, Janka Biznárová, et al. 2023. Transmon qubit readout fidelity at the threshold for quantum error correction without a quantum-limited amplifier. *npj Quantum Information* 9, 1 (2023), 26. DOI: https://doi.org/10.1038/s41534-023-00689-6

[28] Shifu Chen, Yaru Chen, Zhouyang Wang, Wenjian Qin, Jing Zhang, Heera Nand, Jishuai Zhang, Jun Li, Xiaoni Zhang, Xiaoming Liang, et al. 2023. Efficient sequencing data compression and FPGA acceleration based on a two-step framework. *Frontiers in Genetics* 14 (2023), 1260531.

[29] Yanhao Chen, Yuwei Jin, Fei Hua, Ari Hayes, Ang Li, Yunong Shi, and Eddy Z. Zhang. 2023. A Pulse Generation Framework with Augmented Program-aware Basis Gates and Criticality Analysis. In *Proceedings of the 2023 IEEE International Symposium on High-Performance Computer Architecture*. 773–786. DOI: https://doi.org/10.1109/HPCA56546.2023.10070990

[30] Zhao-Yun Chen, Qi Zhou, Cheng Xue, Xia Yang, Guang-Can Guo, and Guo-Ping Guo. 2018. 64-qubit quantum circuit simulation. *Science Bulletin* 63, 15 (2018), 964–971. DOI: https://doi.org/10.1016/j.scib.2018.06.007

[31] Sai Pavan Chitta, Tianpu Zhao, Ziwen Huang, Ian Mondragon-Shem, and Jens Koch. 2022. Computer-aided quantization and numerical analysis of superconducting circuits. *New Journal of Physics* 24, 10 (2022), 103020.

[32] J. Ignacio Cirac, David Perez-Garcia, Norbert Schuch, and Frank Verstraete. 2021. Matrix product states and projected entangled pair states: Concepts, symmetries, theorems. *Reviews of Modern Physics* 93, 4 (2021), 045003.

[33] Sean Clark, Richard Jozsa, and Noah Linden. 2008. Generalized clifford groups and simulation of associated quantum circuits. *Quantum Information & Computation* 8, 1 (2008), 106–126.

[34] John Clyne, Pablo Mininni, Alan Norton, and Mark Rast. 2007. Interactive desktop analysis of high resolution simulations: Application to turbulent plume dynamics and current sheet formation. *New Journal of Physics* 9, 8 (2007), 301. DOI: https://doi.org/10.1088/1367-2630/9/8/301

[35] Jason Cong, Zhenman Fang, Michael Lo, Hanrui Wang, Jingxian Xu, and Shaochong Zhang. 2018. Understanding performance differences of FPGAs and GPUs. In *Proceedings of the 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 93–96.

[36] Dassault Systèmes. 2025. SOLIDWORKS: solid modeling computer-aided design (CAD) and computer-aided engineering (CAE) software. Dassault Systèmes, Vélizy-Villacoublay, France.

[37] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. 2019. On the qubit routing problem. In *Proceedings of the 14th Conference on the Theory of Quantum Computation, Communication and Cryptography*. W. van Dam and L. Mancinska (Eds.), Vol. 135, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 5:1–5:32. DOI: https://doi.org/10.4230/LIPIcs.TQC.2019.5

[38] Alexander M. Dalzell, Sam McArdle, Mario Berta, Przemyslaw Bienias, Chi-Fang Chen, András Gilyén, Connor T. Hann, Michael J. Kastoryano, Emil T. Khabiboulline, Aleksander Kubica, et al. 2023. Quantum algorithms: A survey of applications and end-to-end complexities. (2023). arXiv:2310.03011. Retrieved from https://arxiv.org/abs/2310.03011

[39] Poulami Das, Eric Kessler, and Yunong Shi. 2023. The imitation game: Leveraging copycats for robust native gate selection in NISQ programs. In *Proceedings of the 2023 IEEE International Symposium on High-Performance Computer Architecture*. 787–801. DOI: https://doi.org/10.1109/HPCA56546.2023.10071025

[40] F. Pelayo García de Arquer, Dmitri V. Talapin, Victor I. Klimov, Yasuhiko Arakawa, Manfred Bayer, and Edward H. Sargent. 2021. Semiconductor quantum dots: Technological progress and future challenges. *Science* 373, 6555 (2021), eaaz8541. DOI: https://doi.org/10.1126/science.aaz8541 arXiv:https://www.science.org/doi/pdf/10.1126/science.aaz8541

[41] A. B. de Avila, R. H. S. Reiser, M. L. Pilla, and A. C. Yamin. 2020. State-of-the-art quantum computing simulators: Features, optimizations, and improvements for D-GM. *Neurocomputing* 393 (2020), 223–233. DOI: https://doi.org/10.1016/j.neucom.2019.01.118

[42] J. de Fine Licht, C. A. Pattison, A. Ziogas, D. Simmons-Duffin, and T. Hoefler. 2022. Fast arbitrary precision floating point on FPGA. In *Proceedings of the 2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE Computer Society, Los Alamitos, CA, USA, 1–9. DOI: https://doi.org/10.1109/FCCM53951.2022.9786219

[43] Hans De Raedt, Fengping Jin, Dennis Willsch, Madita Willsch, Naoki Yoshioka, Nobuyasu Ito, Shengjun Yuan, and Kristel Michielsen. 2019. Massively parallel quantum computer simulator, eleven years later. *Computer Physics Communications* 237 (2019), 47–61. DOI: https://doi.org/10.1016/j.cpc.2018.11.005

[44] K. De Raedt, K. Michielsen, H. De Raedt, B. Trieu, G. Arnold, M. Richter, Th. Lippert, H. Watanabe, and N. Ito. 2007. Massively parallel quantum computer simulator. *Computer Physics Communications* 176, 2 (2007), 121–136. DOI: https://doi.org/10.1016/j.cpc.2006.08.007

[45] Rina Dechter. 1999. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence* 113, 1 (1999), 41–85. DOI: https://doi.org/10.1016/S0004-3702(99)00059-4

[46] R. Delaney, M. Urmey, S. Mittal, B. Brubaker, J. Kindem, P. Burns, C. Regal, and K. Lehnert. 2022. Superconducting-qubit readout via low-backaction electro-optic transduction. *Nature* 606, 06 (2022), 489–493. DOI: https://doi.org/10.1038/s41586-022-04720-2

[47] P. Deutsch. 1996. GZIP file format specification version 4.3. RFC 1952.

[48] Jun Doi and Hiroshi Horii. 2020. Cache blocking technique to large scale quantum computing simulation on supercomputers. In *Proceedings of the 2020 IEEE International Conference on Quantum Computing and Engineering*. IEEE, 212–222.

[49] Jun Doi, Hiroshi Horii, and Christopher Wood. 2023. Efficient techniques to GPU accelerations of multi-shot quantum computing simulations. (2023). arXiv:2308.03399. Retrieved from https://arxiv.org/abs/2308.03399
[50] Jun Doi, Hitomi Takahashi, Rudy Raymond, Takashi Imamichi, and Hiroshi Horii. 2019. Quantum computing simulator on a heterogenous HPC system. In *Proceedings of the 16th ACM International Conference on Computing Frontiers* . Association for Computing Machinery, New York, NY, USA, 85–93. DOI: https://doi.org/10.1145/3310273.3323053
[51] Jack Dongarra, Jeffrey Hittinger, John Bell, Luis Chacon, Robert Falgout, Michael Heroux, Paul Hovland, Esmond Ng, Clayton Webster, and Stefan Wild. 2014. *Applied Mathematics Research for Exascale Computing*. Technical Report. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States).
[52] Menghan Dou, Tianrui Zou, Yuan Fang, Jing Wang, Dongyi Zhao, Lei Yu, Boying Chen, Wenbo Guo, Ye Li, Zhaoyun Chen, et al. 2022. QPanda: High-performance quantum computing framework for multiple application scenarios. (2022). arXiv:2212.14201. Retrieved from https://arxiv.org/abs/2212.14201
[53] Adam Dziekonski, Adam Lamecki, and Michal Mrozowski. 2011. GPU acceleration of multilevel solvers for analysis of microwave components with finite element method. *IEEE Microwave and Wireless Components Letters* 21, 1 (2011), 1–3. DOI: https://doi.org/10.1109/LMWC.2010.2089974
[54] Stavros Efthymiou, Marco Lazzarin, Andrea Pasquale, and Stefano Carrazza. 2022. Quantum simulation with just-in-time compilation. *Quantum* 6 (2022), 814. DOI: https://doi.org/10.22331/q-2022-09-22-814
[55] Stavros Efthymiou, Sergi Ramos-Calderer, Carlos Bravo-Prieto, Adrián Pérez-Salinas, Diego García-Martín, Artur Garcia-Saez, José Ignacio Latorre, and Stefano Carrazza. 2021. Qibo: a framework for quantum simulation with hardware acceleration. *Quantum Science and Technology* 7, 1 (2021), 015018. DOI: https://doi.org/10.1088/2058-9565/ac39f5
[56] Glen Evenbly. 2022. A practical guide to the numerical implementation of tensor networks i: Contractions, decompositions, and gauge freedom. *Frontiers in Applied Mathematics and Statistics* 8 (2022), 806549.
[57] Oak Ridge Leadership Computing Facility. 2024. Summit. (2024). Retrieved October 10, 2024 from https://www.olcf.ornl.gov/summit/
[58] Bo Fang, M. Yusuf Özkaya, Ang Li, Ümit V Çatalyürek, and Sriram Krishnamoorthy. 2022. Efficient hierarchical state vector simulation of quantum circuits via acyclic graph partitioning. In *Proceedings of the 2022 IEEE International Conference on Cluster Computing*. IEEE, 289–300.
[59] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A quantum approximate optimization algorithm. (2014). arXiv:1411.4028. Retrieved from https://arxiv.org/abs/1411.4028
[60] Aneeqa Fatima and Igor L. Markov. 2021. Faster Schrödinger-style simulation of quantum circuits. In *Proceedings of the 2021 IEEE International Symposium on High-Performance Computer Architecture*. 194–207. DOI: https://doi.org/10.1109/HPCA51647.2021.00026
[61] Jiri Filipovic, Igor Peterlik, and Jan Fousek. 2009. GPU acceleration of equations assembly in finite elements method-preliminary results. In *Proceedings of the SAAHPC: Symposium on Application Accelerators in HPC*.
[62] Matthew P. A. Fisher, Vedika Khemani, Adam Nahum, and Sagar Vijay. 2023. Random quantum circuits. *Annual Review of Condensed Matter Physics* 14, 1 (2023), 335–379. DOI: https://doi.org/10.1146/annurev-conmatphys-031720-030658 arXiv:https://doi.org/10.1146/annurev-conmatphys-031720-030658
[63] Simon Pettersson Fors, Jorge Fernández-Pendás, and Anton Frisk Kockum. 2024. Comprehensive explanation of ZZ coupling in superconducting qubits. (2024). arXiv:2408.15402. Retrieved from https://arxiv.org/abs/2408.15402
[64] Michael Foss-Feig, Guido Pagano, Andrew C. Potter, and Norman Y. Yao. 2024. Progress in trapped-ion quantum simulation. *Annual Review of Condensed Matter Physics* 16 (2024).
[65] Jay M. Gambetta, Jerry M. Chow, and Matthias Steffen. 2017. Building logical qubits in a superconducting quantum computing system. *npj Quantum Information* 3, 1 (2017), 2.
[66] I. M. Georgescu, S. Ashhab, and Franco Nori. 2014. Quantum simulation. *Reviews of Modern Physics* 86, 1 (2014), 153–185. DOI: https://doi.org/10.1103/RevModPhys.86.153
[67] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2022. A survey of quantization methods for efficient neural network inference. In *Proceedings of the Low-Power Computer Vision*. Chapman and Hall/CRC, 291–326.
[68] Craig Gidney. 2021. Stim: A fast stabilizer circuit simulator. *Quantum* 5 (2021), 497.
[69] Vibhav Gogate and Rina Dechter. 2004. A complete anytime algorithm for treewidth. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*. 201–208.
[70] David Gosset, Daniel Grier, Alex Kerzner, and Luke Schaeffer. 2024. Fast simulation of planar Clifford circuits. *Quantum* 8 (2024), 1251.
[71] Daniel Gottesman. 1998. The Heisenberg representation of quantum computers. In *Proc. XXII International Colloquium on Group Theoretical Methods in Physics*. 32–43.
[72] Johnnie Gray and Stefanos Kourtis. 2021. Hyper-optimized tensor network contraction. *Quantum* 5 (2021), 410. DOI: https://doi.org/10.22331/q-2021-03-15-410

[73] Peter Groszkowski and Jens Koch. 2021. Scqubits: A Python package for superconducting qubits. *Quantum* 5 (2021), 583. DOI: https://doi.org/10.22331/q-2021-11-17-583

[74] L. K. Grover. 1997. Quantum mechanics helps in searching for a needle in a haystack. *Physical Review Letters* 79, 2 (1997), 325. https://doi.org/10.1103/PhysRevLett.79.325

[75] Xiu Gu, Anton Frisk Kockum, Adam Miranowicz, Yu xi Liu, and Franco Nori. 2017. Microwave photonics with superconducting quantum circuits. *Physics Reports* 718-719 (2017), 1–102. DOI: https://doi.org/10.1016/j.physrep.2017.10.002 Microwave photonics with superconducting quantum circuits.

[76] Gian Giacomo Guerreschi, Justin Hogaboam, Fabio Baruffa, and Nicolas P. D. Sawaya. 2020. Intel Quantum Simulator: A cloud-ready high-performance simulator of quantum circuits. *Quantum Science and Technology* 5, 3 (2020), 034007. DOI: https://doi.org/10.1088/2058-9565/ab8505

[77] Pierre Guilmin, Ronan Gautier, Adrien Bocquet, and Élie Genois. 2024. dynamiqs: An open-source Python library for GPU-accelerated and differentiable simulations of quantum systems. (2024). Retrieved from https://github.com/dynamiqs/dynamiqs

[78] Eladio Gutierrez, Sergio Romero, Maria A. Trenas, and Emilio L. Zapata. 2007. Simulation of quantum gates on a novel GPU architecture. In *Proceedings of the 7th Conference on 7th WSEAS International Conference on Systems Theory and Scientific Computation.* World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 121–126.

[79] Eladio Gutiérrez, Sergio Romero, María A. Trenas, and Emilio L. Zapata. 2010. Quantum computer simulation using the CUDA programming model. *Computer Physics Communications* 181, 2 (2010), 283–300. DOI: https://doi.org/10.1016/j.cpc.2009.09.021

[80] Kamil Halbiniak, Krzysztof Rojek, Sergio Iserte, and Roman Wyrzykowski. 2024. Unleashing the potential of mixed precision in AI-accelerated CFD simulation on intel CPU/GPU architectures. In *Proceedings of the Computational Science.* Leonardo Franco, Clélia de Mulatier, Maciej Paszynski, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M. A. Sloot (Eds.), Springer Nature Switzerland, Cham, 203–217.

[81] Thomas Häner and Damian S. Steiger. 2017. 0.5 petabyte simulation of a 45-qubit quantum circuit. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.* 1–10.

[82] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, et al. 2020. Array programming with NumPy. *Nature* 585, 7825 (2020), 357–362. DOI: https://doi.org/10.1038/s41586-020-2649-2

[83] Stijn Heldens, Pieter Hijma, Ben Van Werkhoven, Jason Maassen, Adam S. Z. Belloum, and Rob V. Van Nieuwpoort. 2020. The landscape of exascale research: A data-driven literature analysis. *ACM Computing Surveys* 53, 2 (2020), 1–43.

[84] Sengthai Heng, Taekyung Kim, and Youngsun Han. 2020. Exploiting GPU-based Parallelism for Quantum Computer Simulation: A Survey. *IEIE Transactions on Smart Processing and Computing* 9, 12 (2020), 468–476. DOI: https://doi.org/10.5573/IEIESPC.2020.9.6.468

[85] Loïc Henriet, Lucas Beguin, Adrien Signoles, Thierry Lahaye, Antoine Browaeys, Georges-Olivier Reymond, and Christophe Jurczak. 2020. Quantum computing with neutral atoms. *Quantum* 4 (2020), 327.

[86] Nhut-Minh Ho, Himeshi De silva, and Weng-Fai Wong. 2021. GRAM: A framework for dynamically mixing precisions in GPU applications. *ACM Transactions on Architecture and Code Optimization* 18, 2 (2021), 1–24.

[87] Yunpyo Hong, Seokhun Jeon, Sihyeong Park, and Byung-Soo Kim. 2022. Quantum circuit simulator based on FPGA. In *Proceedings of the 2022 13th International Conference on Information and Communication Technology Convergence* . 1909–1911. DOI: https://doi.org/10.1109/ICTC55196.2022.9952408

[88] Tsung-Wei Huang. 2023. qTask: Task-parallel quantum circuit simulation with incrementality. In *Proceedings of the 2023 IEEE International Parallel and Distributed Processing Symposium.* IEEE, 746–756.

[89] Xuanteng Huang, Xianwei Zhang, Panfei Yang, and Nong Xiao. 2023. Benchmarking GPU tensor cores on general matrix multiplication kernels through CUTLASS. *Applied Sciences* 13, 24 (2023), 13022.

[90] Ansys INC. 2025. Ansys Electronics Desktop: electromagnetic, signal integrity, thermal and electro-mechanical simulations tool. Canonsburg, USA. Retrieved September 24, 2025 from https://www.ansys.com

[91] Nanoacademic Technologies Inc. 2023. QTCAD. (Jul 2023). Retrieved from https://nanoacademic.com/solutions/qtcad/

[92] SunMagnetics Inc. 2024. InductEX. Stellenbosch, South Africa. Retrieved October 10, 2024 from https://www.sunmagnetics.com/product/inductex

[93] Amit Jamadagni Gangapuram, Andreas Läuchli, and Cornelius Hempel. 2024. Benchmarking quantum computer simulation software packages: State vector simulators. *SciPost Physics Core* 7, 4 (2024), 075.

[94] J. R. Johansson, P. D. Nation, and Franco Nori. 2012. QuTiP: An open-source Python framework for the dynamics of open quantum systems. *Computer Physics Communications* 183, 8 (2012), 1760–1772. DOI: https://doi.org/10.1016/j.cpc.2012.02.021

[95] J. R. Johansson, P. D. Nation, and Franco Nori. 2013. QuTiP 2: A Python framework for the dynamics of open quantum systems. *Computer Physics Communications* 184, 4 (2013), 1234–1240. DOI: https://doi.org/10.1016/j.cpc.2012.11.019

[96] Tyson Jones, Anna Brown, Ian Bush, and Simon C. Benjamin. 2019. QuEST and high performance simulation of quantum computers. *Scientific Reports* 9, 1 (2019). DOI: https://doi.org/10.1038/s41598-019-47174-9

[97] Tyson Jones, Bálint Koczor, and Simon C. Benjamin. 2023. Distributed simulation of statevectors and density matrices. (2023). arXiv:2311.01512. Retrieved from https://arxiv.org/abs/2311.01512

[98] Norman P. Jouppi, Cliff Young, Nishant Patil, and David Patterson. 2018. A domain-specific architecture for deep neural networks. *Communications of the ACM* 61, 9 (2018), 50–59.

[99] Yaninee Jungjarassub and Krerk Piromsopa. 2022. A performance optimization of quantum computing simulation using FPGA. In *Proceedings of the 2022 19th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*. 1–4. DOI: https://doi.org/10.1109/ECTI-CON54298.2022.9795571

[100] Pradnya Khalate, Xin-Chuan Wu, Shavindra Premaratne, Justin Hogaboam, Adam Holmes, Albert Schmitz, Gian Giacomo Guerreschi, Xiang Zou, and Anne Y. Matsuura. 2022. An LLVM-based C++ compiler toolchain for variational hybrid quantum-classical algorithms and quantum accelerators. (2022). arXiv:2202.11142. Retrieved from https://arxiv.org/abs/2202.11142

[101] A. U. Khalid, Z. Zilic, and K. Radecka. 2004. FPGA emulation of quantum circuits. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*. 310–315. DOI: https://doi.org/10.1109/ICCD.2004.1347938

[102] David Kielpinski, C. R. Monroe, and D. J. Wineland. 2002. Architecture for a large-scale ion-trap quantum computer. *Nature* 417, 6890 (2002), 709–11. DOI: https://doi.org/10.1038/nature00784

[103] Youngseok Kim, Andrew Eddins, Sajant Anand, Ken Xuan Wei, Ewout Van Den Berg, Sami Rosenblatt, Hasan Nayfeh,Yantao Wu, Michael Zaletel, Kristan Temme, et al. 2023. Evidence for the utility of quantum computing before fault tolerance. *Nature* 618, 7965 (2023), 500–505.

[104] Morten Kjaergaard, Mollie E. Schwartz, Jochen Braumüller, Philip Krantz, Joel I.-J. Wang, Simon Gustavsson, and William D. Oliver. 2020. Superconducting qubits: Current state of play. *Annual Review of Condensed Matter Physics* 11, 1 (2020), 369–395.

[105] Prasad Sarangapani, James Fonseca, Daniel F. Mejia, James Charles, Woody Gilbertson, Tarek Ahmed Ameen, Hesameddin Ilatikhameneh, Andrew Roché, Lars Bjaalie, Sebastian Steiger, David Ebert, Matteo Mannino, Hong-Hyun Park, Tillmann Christoph Kubis, Michael Povolotskyi, Michael McLennan, and Gerhard Klimeck. 2005. Quantum Dot Lab. https://doi.org/doi:10.21981/CPJ9-3Q95

[106] Gerhard Klimeck, Marek Korkusinski, Haiying Xu, Seungwon Lee, Sebastien Goasguen, and Faisal Saied. 2005. Building and deploying community nanotechnology software tools on nanoHUB. org-atomistic simulations of multimillion-atom quantum dot nanostructures. In *Proceedings of the 5th IEEE Conference on Nanotechnology*. IEEE.

[107] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, et al. 2008. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep* 15 (2008), 181.

[108] Pieter Kok. 2016. Photonic quantum information processing. *Contemporary Physics* 57, 4 (2016), 526–544.

[109] Sandoko Kosen, Hang-Xi Li, Marcus Rommel, Daryoush Shiri, Christopher Warren, Leif Grönberg, Jaakko Salonen, Tahereh Abad, Janka Biznárová, Marco Caputo, et al. 2022. Building blocks of a flip-chip integrated superconducting quantum processor. *Quantum Science and Technology* 7, 3 (2022), 035018. DOI: https://doi.org/10.1088/2058-9565/ac734b

[110] Sebastian Krämer, David Plankensteiner, Laurin Ostermann, and Helmut Ritsch. 2018. QuantumOptics. jl: A Julia framework for simulating open quantum systems. *Computer Physics Communications* 227 (2018), 109–116.

[111] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver. 2019. A quantum engineer's guide to superconducting qubits. *Applied Physics Reviews* 6, 2 (2019), 021318. DOI: https://doi.org/10.1063/1.5089550 arXiv:https://pubs.aip.org/aip/apr/article-pdf/doi/10.1063/1.5089550/16667201/021318_1_online.pdf

[112] S. Krinner, S. Storz, P. Kurpiers, P. Magnard, J. Heinsoo, R. Keller, J. Lütolf, C. Eichler, and A. Wallraff. 2019. Engineering cryogenic setups for 100-qubit scale superconducting circuit systems. *EPJ Quantum Technology* 6, 1 (2019), 2. DOI: https://doi.org/10.1140/epjqt/s40507-019-0072-0

[113] Argonne National Lab. 2017. Introducing Argonne's Theta Supercomputer. (2017). Retrieved from https://www.osti.gov/biblio/1371569

[114] Lawrence Livermore National Laboratory. 2024. Sierra. (2024). Retrieved October 10, 2024 from https://hpc.llnl.gov/hardware/compute-platforms/sierra

[115] Sriram Lakshminarasimhan, Neil Shah, Stephane Ethier, Seung-Hoe Ku, C. S. Chang, Scott Klasky, Rob Latham, Rob Ross, and Nagiza F. Samatova. 2013. ISABELA for effective in situ compression of scientific data. *Concurrency and Computation: Practice and Experience* 25, 4 (2013), 524–540. DOI: https://doi.org/10.1002/cpe.2887 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.2887

[116] Joseph K. L. Lee, Maurice Jamieson, Nick Brown, and Ricardo Jesus. 2023. Test-driving RISC-V vector hardware for HPC. In *Proceedings of the High Performance Computing*. Amanda Bienz, Michèle Weiland, Marc Baboulin, and Carola Kruse (Eds.), Springer Nature Switzerland, Cham, 419–432.

[117] Ang Li, Bo Fang, Christopher Granade, Guen Prawiroatmodjo, Bettina Heim, Martin Roetteler, and Sriram Krishnamoorthy. 2021. Sv-sim: Scalable pgas-based state vector simulation of quantum circuits. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14.

[118] Ang Li, Omer Subasi, Xiu Yang, and Sriram Krishnamoorthy. 2020. Density matrix quantum circuit simulation via the BSP machine on modern GPU clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15. DOI: https://doi.org/10.1109/SC41405.2020.00017

[119] Boxi Li, Shahnawaz Ahmed, Sidhant Saraogi, Neill Lambert, Franco Nori, Alexander Pitchford, and Nathan Shammah. 2022. Pulse-level noisy quantum circuits with QuTiP. *Quantum* 6 (Jan. 2022), 630. https://doi.org/10.22331/q-2022-01-24-630

[120] Gushu Li, Yufei Ding, and Yuan Xie. 2019. Tackling the qubit mapping problem for NISQ-Era quantum devices. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, New York, NY, USA, 1001–1014. DOI: https://doi.org/10.1145/3297858.3304023

[121] Riling Li, Bujiao Wu, Mingsheng Ying, Xiaoming Sun, and Guangwen Yang. 2020. Quantum supremacy circuit simulation on sunway taihulight. *IEEE Transactions on Parallel and Distributed Systems* 31, 4 (2020), 805–816. DOI: https://doi.org/10.1109/TPDS.2019.2947511

[122] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Shaomeng Li, Hanqi Guo, Zizhong Chen, and Franck Cappello. 2018. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *Proceedings of the 2018 IEEE International Conference on Big Data*. 438–447. DOI: https://doi.org/10.1109/BigData.2018.8622520

[123] Peter Lindstrom. 2014. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2674–2683. DOI: https://doi.org/10.1109/TVCG.2014.2346458

[124] Peter Lindstrom and Martin Isenburg. 2006. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics* 12, 5 (2006), 1245–1250. DOI: https://doi.org/10.1109/TVCG.2006.143

[125] Ji Liu, Alvin Gonzales, Benchen Huang, Zain Hamid Saleem, and Paul Hovland. 2024. QuCLEAR: clifford extraction and absorption for significant reduction in quantum circuit size. arXiv:2408.13316. Retrieved from https://arxiv.org/abs/2408.13316

[126] Lei Liu and Xinglei Dou. 2021. QuCloud: A new qubit mapping mechanism for multi-programming quantum computing in cloud environment. In *Proceedings of the 2021 IEEE International Symposium on High-Performance Computer Architecture*. 167–178. DOI: https://doi.org/10.1109/HPCA51647.2021.00024

[127] Yong Liu, Xin Liu, Fang Li, Haohuan Fu, Yuling Yang, Jiawei Song, Pengpeng Zhao, Zhen Wang, Dajia Peng, Huarong Chen, et al. 2021. Closing the "quantum supremacy" gap: Achieving real-time simulation of a random quantum circuit using a new sunway supercomputer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[128] Dmitry I. Lyakh. 2015. An efficient tensor transpose algorithm for multicore CPU, Intel Xeon Phi, and NVidia Tesla GPU. *Computer Physics Communications* 189 (2015), 84–91. DOI: https://doi.org/10.1016/j.cpc.2014.12.013

[129] Danylo Lykov, Angela Chen, Huaxuan Chen, Kristopher Keipert, Zheng Zhang, Tom Gibbs, and Yuri Alexeev. 2021. Performance evaluation and acceleration of the QTensor quantum circuit simulator on GPUs. In *Proceedings of the 2021 IEEE/ACM Second International Workshop on Quantum Computing Software*. 27–34. DOI: https://doi.org/10.1109/QCS54837.2021.00007

[130] Danylo Lykov, Roman Schutski, Alexey Galda, Valeri Vinokur, and Yuri Alexeev. 2022. Tensor network quantum simulator with step-dependent parallelization. In *Proceedings of the 2022 IEEE International Conference on Quantum Computing and Engineering*. IEEE, 582–593.

[131] Lars S. Madsen, Fabian Laudenbach, Mohsen Falamarzi Askarani, Fabien Rortais, Trevor Vincent, Jacob F. F. Bulmer, Filippo M. Miatto, Leonhard Neuhaus, Lukas G. Helt, Matthew J. Collins, et al. 2022. Quantum computational advantage with a programmable photonic processor. *Nature* 606, 7912 (2022), 75–81. https://doi.org/10.1038/s41586-022-04725-x

[132] Naveed Mahmud and Esam El-Araby. 2018. A scalable high-precision and high-throughput architecture for emulation of quantum algorithms. In *Proceedings of the 2018 31st IEEE International System-on-Chip Conference*. 206–212. DOI: https://doi.org/10.1109/SOCC.2018.8618545

[133] Naveed Mahmud and Esam El-Araby. 2018. A scalable high-precision and high-throughput architecture for emulation of quantum algorithms. In *2018 31st IEEE International System-on-Chip Conference (SOCC)*. 206–212. https://doi.org/10.1109/SOCC.2018.8618545

[134] Salvatore Mandrà, Jeffrey Marshall, Eleanor G. Rieffel, and Rupak Biswas. 2021. HybridQ: A hybrid simulator for quantum circuits. In *Proceedings of the 2021 IEEE/ACM Second International Workshop on Quantum Computing Software*. IEEE, 99–109.

[135] Igor L. Markov, Aneeqa Fatima, Sergei V. Isakov, and Sergio Boixo. 2020. Massively parallel approximate simulation of hard quantum circuits. In *Proceedings of the 2020 57th ACM/IEEE Design Automation Conference*. 1–6. DOI: https://doi.org/10.1109/DAC18072.2020.9218591

[136] Igor L. Markov and Yaoyun Shi. 2008. Simulating quantum computation by contracting tensor networks. *SIAM J. Comput.* 38, 3 (2008), 963–981. DOI: https://doi.org/10.1137/050644756 arXiv:https://doi.org/10.1137/050644756

[137] Sergi Masot-Llima and Artur Garcia-Saez. 2024. Stabilizer tensor networks: Universal quantum simulator on a basis of stabilizer states. *Physical Review Letters* 133, 23 (2024), 230601.

[138] Satvik Maurya, Chaithanya Naik Mude, William D. Oliver, Benjamin Lienhard, and Swamit Tannu. 2023. Scaling qubit readout with hardware efficient machine learning architectures. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–13.

[139] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C. Benjamin, and Xiao Yuan. 2020. Quantum computational chemistry. *Reviews of Modern Physics* 92, 1 (2020), 015003. DOI: https://doi.org/10.1103/RevModPhys.92.015003

[140] Inc. Meta Platforms. Zstandard - Fast real-time compression algorithm. (n.d.). Retrieved from https://github.com/facebook/zstd

[141] Ashley Montanaro. 2016. Quantum algorithms: An overview. *npj Quantum Information* 2, 1 (2016), 1–8.

[142] José E. Moreira, Kit Barton, Steven Battle, Peter Bergner, Ramon Bertran, Puneeth Bhat, Pedro Caldeira, David Edelsohn, Gordon Fossum, Brad Frey, et al. 2021. A matrix math facility for Power ISA (TM) processors. (2021). arXiv:2104.03142. Retrieved from https://arxiv.org/abs/2104.03142

[143] Michael A. Nielsen and Isaac L. Chuang. 2023. *Quantum Computation and Quantum Information*. Cambridge University Press.

[144] ROYUD Nishino and Shohei Hido Crissman Loomis. 2017. Cupy: A numpy-compatible library for nvidia gpu calculations. *31st Confernce on Neural Information Processing Systems* 151, 7 (2017).

[145] Srikrishna Omkar, Seok-Hyung Lee, Yong Siah Teo, Seung-Woo Lee, and Hyunseok Jeong. 2022. All-photonic architecture for scalable quantum computing with greenberger-horne-zeilinger states. *PRX Quantum* 3, 3 (2022), 030309. DOI: https://doi.org/10.1103/PRXQuantum.3.030309

[146] Alexandru Paler, Alwin Zulehner, and Robert Wille. 2021. NISQ circuit compilation is the travelling salesman problem on a torus. *Quantum Science and Technology* 6, 2 (2021), 025016. https://doi.org/10.1088/2058-9565/abe665

[147] Daeyoung Park, Heehoon Kim, Jinpyo Kim, Taehyun Kim, and Jaejin Lee. 2022. SnuQS: Scaling quantum circuit simulation using storage devices. In *Proceedings of the 36th ACM International Conference on Supercomputing*. Association for Computing Machinery, New York, NY, USA, 13 pages. DOI: https://doi.org/10.1145/3524059.3532375

[148] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Curran Associates Inc., Red Hook, NY, USA.

[149] Tirthak Patel, Daniel Silver, and Devesh Tiwari. 2022. Geyser: A compilation framework for quantum computing with neutral atoms. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. Association for Computing Machinery, New York, NY, USA, 383–395. DOI: https://doi.org/10.1145/3470496.3527428

[150] Bishnu Patra, Jeroen P. G. van Dijk, Sushil Subramanian, Andrea Corna, Xiao Xue, Charles Jeon, Farhana Sheikh, Esdras Juarez-Hernandez, Brando Perez Esparza, Huzaifa Rampurawala, et al. 2020. 19.1 a scalable cryo-CMOS 2-to-20GHz digitally intensive controller for 4×32 frequency multiplexed spin qubits/transmons in 22nm FinFET technology for quantum computers. In *Proceedings of the 2020 IEEE International Solid-State Circuits Conference*. 304–306. DOI: https://doi.org/10.1109/ISSCC19947.2020.9063109

[151] Taylor L. Patti, Jean Kossaifi, Anima Anandkumar, and Susanne F. Yelin. 2022. Variational quantum optimization with multibasis encodings. *Physical Review Research* 4, 3 (2022), 033142. DOI: https://doi.org/10.1103/PhysRevResearch.4.033142

[152] Taylor L. Patti, Jean Kossaifi, Susanne F. Yelin, and Anima Anandkumar. 2021. Tensorly-quantum: Quantum machine learning with tensor methods. (2021). arXiv:2112.10239. Retrieved from https://arxiv.org/abs/2112.10239

[153] Taylor L. Patti, Omar Shehab, Khadijeh Najafi, and Susanne F. Yelin. 2022. Markov chain Monte Carlo enhanced variational quantum algorithms. *Quantum Science and Technology* 8, 1 (2022), 015019.

[154] Edwin Pednault, John A. Gunnels, Giacomo Nannicini, L. Haoresh, Thomas Magerlein, Edgar Solomonik, Erik W Draeger, Eric T. Holland, and Robert Wisnieff. 2018. *Breaking the 49-qubit Barrier in the Simulation of Quantum Circuits*. Technical Report. Lawrence Livermore National Laboratory (LLNL), Livermore, CA (United States).

[155] Simon Pettersson Fors, Linus von Ekensteen Löfgren, Nikita Suprun, Patric Holmvall, Pontus Vikstål, Göran Johansson, Anton Frisk Kockum, and Jorge Fernández-Pendás. 2024. CSQR: Chalmers Superconducting Qubit Repository. (2024).

[156] Bagas Prabowo, Guoji Zheng, Mohammadreza Mehrpoo, Bishnu Patra, Patrick Harvey-Collard, Jurgen Dijkema, Amir Sammak, Giordano Scappucci, Edoardo Charbon, Fabio Sebastiano, et al. 2021. 13.3 A 6-to-8GHz 0.17mW/Qubit Cryo-CMOS receiver for multiple spin qubit readout in 40nm CMOS technology. In *Proceedings of the 2021 IEEE International Solid-State Circuits Conference*. 212–214. DOI: https://doi.org/10.1109/ISSCC42613.2021.9365848

[157] John Preskill. 2018. Quantum computing in the NISQ era and beyond. *Quantum* 2 (2018), 79.

[158] Gadi Aleksandrowicz et al. 2019. Qiskit: An Open-source Framework for Quantum Computing. Zenodo. https://doi.org/10.5281/zenodo.2573505

[159] Christopher J. K. Richardson, Vincenzo Lordi, Shashank Misra, and Javad Shabani. 2020. Materials science for quantum information science and technology. *MRS Bulletin* 45, 6 (2020), 485–497. DOI: https://doi.org/10.1557/mrs.2020.147

[160] Jacquiline Romero and Gerard Milburn. 2024. Photonic quantum computing. arXiv:2404.03367. Retrieved from https://arxiv.org/abs/2404.03367

[161] Vahid Sarvestan, Hamid Reza Mirdamadi, and Mostafa Ghayour. 2017. Vibration analysis of cracked Timoshenko beam under moving load with constant velocity and acceleration by spectral finite element method. *International Journal of Mechanical Sciences* 122 (2017), 318–330. DOI: https://doi.org/10.1016/j.ijmecsci.2017.01.035

[162] Naoto Sasaki, Kento Sato, Toshio Endo, and Satoshi Matsuoka. 2015. Exploration of lossy compression for application-level checkpoint/restart. In *Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE Computer Society, USA, 914–922. DOI: https://doi.org/10.1109/IPDPS.2015.67

[163] Ulrich Schollwöck. 2011. The density-matrix renormalization group in the age of matrix product states. *Annals of physics* 326, 1 (2011), 96–192.

[164] Milan Shah, Xiaodong Yu, Sheng Di, Danylo Lykov, Yuri Alexeev, Michela Becchi, and Franck Cappello. 2023. GPU-accelerated error-bounded compression framework for quantum circuit simulations. In *2404.033672023 IEEE International Parallel and Distributed Processing Symposium*. 757–767. DOI: https://doi.org/10.1109/IPDPS54959.2023.00081

[165] Ahmad Husni Mohd Shapri, Norazeani Abdul Rahman, Syed Muhammad Mamduh Syed Zakaria, Kiu Kwong Chieh, and Shakir Saat. 2024. Optimization and analysis of FPGA-based systolic array for matrix multiplication. *AIP Conference Proceedings* 2898, 1 (2024), 030007. DOI: https://doi.org/10.1063/5.0192098 arXiv:https://pubs.aip.org/aip/acp/article-pdf/doi/10.1063/5.0192098/19579660/030007_1_5.0192098.pdf

[166] P. W. Shor. 1994. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*. IEEE Comput. Soc. Press, Washington, DC, USA, 124. DOI: https://doi.org/10.1109/SFCS.1994.365700

[167] Peter W. Shor. 1995. Scheme for reducing decoherence in quantum computer memory. *Physical Review A* 52, 4 (1995), R2493–R2496. DOI: https://doi.org/10.1103/PhysRevA.52.R2493

[168] Irfan Siddiqi. 2021. Engineering high-coherence superconducting qubits. *Nature Reviews Materials* 6, 10 (2021), 875–891. DOI: https://doi.org/10.1038/s41578-021-00370-4

[169] Cristina Silvano, Daniele Ielmini, Fabrizio Ferrandi, Leandro Fiorin, Serena Curzel, Luca Benini, Francesco Conti, Angelo Garofalo, Cristian Zambelli, Enrico Calore, et al. 2025. A survey on deep learning hardware accelerators for heterogeneous HPC platforms. *Comput. Surveys* 57, 11 (2025), 1–39.

[170] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Caroline Collange, and Fernando Magno Quintao Pereira. 2018. Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. ACM, New York, NY, USA, 113–125. DOI: https://doi.org/10.1145/3168822

[171] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. 2021. t|ket⟩: a retargetable compiler for NISQ devices. *Quantum Science and Technology* 6 (2021), 014003. DOI: https://doi.org/10.1088/2058-9565/ab8e92

[172] Sergei Slussarenko and Geoff J. Pryde. 2019. Photonic quantum information processing: A concise review. *Applied Physics Reviews* 6, 4 (2019).

[173] Mikhail Smelyanskiy, Nicolas P. D. Sawaya, and Alán Aspuru-Guzik. 2016. qHiPSTER: The quantum high performance software testing environment. (2016). arXiv:1601.07195. Retrieved from https://arxiv.org/abs/1601.07195

[174] Alistair W. R. Smith, Kiran E. Khosla, Chris N. Self, and M. S. Kim. 2021. Qubit readout error mitigation with bit-flip averaging. *Science Advances* 7, 47 (2021), eabi8009. DOI: https://doi.org/10.1126/sciadv.abi8009 arXiv:https://www.science.org/doi/pdf/10.1126/sciadv.abi8009

[175] Kaitlin N. Smith, Michael A. Perlin, Pranav Gokhale, Paige Frederick, David Owusu-Antwi, Richard Rines, Victory Omole, and Frederic Chong. 2023. Clifford-based circuit cutting for quantum simulation. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–13.

[176] Mostafa I. Soliman. 2007. Mat-core: A matrix core extension for general-purpose processors. In *Proceedings of the 2007 International Conference on Computer Engineering and Systems*. 304–310. DOI: https://doi.org/10.1109/ICCES.2007.4447064

[177] Aaron Somoroff, Quentin Ficheux, Raymond A. Mencia, Haonan Xiong, Roman Kuzmin, and Vladimir E. Manucharyan. 2023. Millisecond coherence in a superconducting qubit. *Physical Review Letters* 130, 26 (2023), 267001. DOI: https://doi.org/10.1103/PhysRevLett.130.267001

[178] Damian S. Steiger, Thomas Häner, and Matthias Troyer. 2018. ProjectQ: An open source software framework for quantum computing. *Quantum* 2 (2018), 49. DOI: https://doi.org/10.22331/q-2018-01-31-49

[179] Thomas Strohm, Karen Wintersperger, Florian Dommert, Daniel Basilewitsch, Georg Reuber, Andrey Hoursanov, Thomas Ehmer, Davide Vodola, and Sebastian Luber. 2024. Ion-based quantum computing hardware: Performance and end-user perspective. arXiv:2405.11450. Retrieved from https://arxiv.org/abs/2405.11450

[180] Yasunari Suzuki, Yoshiaki Kawase, Yuya Masumura, Yuria Hiraga, Masahiro Nakadai, Jiabao Chen, Ken M. Nakanishi, Kosuke Mitarai, Ryosuke Imai, Shiro Tamiya, et al. 2021. Qulacs: A fast and versatile quantum circuit simulator for research purpose. *Quantum* 5 (2021), 559. DOI: https://doi.org/10.22331/q-2021-10-06-559

[181] Hisao Tamaki. 2019. Positive-instance driven dynamic programming for treewidth. *Journal of Combinatorial Optimization* 37, 4 (2019), 1283–1311.

[182] Wei Tang, Teague Tomesh, Martin Suchara, Jeffrey Larson, and Margaret Martonosi. 2021. CutQC: Using small Quantum computers for large Quantum circuit evaluations. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM. DOI: https://doi.org/10.1145/3445814.3446758

[183] Archana Tankasala and Hesameddin Ilatikhameneh. 2019. Quantum-kit: Simulating shor's factorization of 24-bit number on desktop. (2019). arXiv:1908.07187. Retrieved from https://arxiv.org/abs/1908.07187

[184] Quantum AI team and collaborators. 2020. qsim. Zenodo. https://doi.org/10.5281/zenodo.4023103

[185] The Blosc Development Team. 2024. BlosC in depth. (2024). Retrieved October 13, 2024 from https://www.blosc.org/pages/blosc-in-depth/

[186] Jiannan Tian, Sheng Di, Chengming Zhang, Xin Liang, Sian Jin, Dazhao Cheng, Dingwen Tao, and Franck Cappello. 2020. Wavesz: A hardware-algorithm co-design of efficient lossy compression for scientific data. In *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 74–88.

[187] Benjamin Villalonga, Sergio Boixo, Bron Nelson, Christopher Henze, Eleanor Rieffel, Rupak Biswas, and Salvatore Mandrà. 2019. A flexible high-performance simulator for verifying and benchmarking quantum circuits implemented on real hardware. *npj Quantum Information* 5, 1 (2019), 86.

[188] Benjamin Villalonga, Dmitry Lyakh, Sergio Boixo, Hartmut Neven, Travis S. Humble, Rupak Biswas, Eleanor G. Rieffel, Alan Ho, and Salvatore Mandrà. 2020. Establishing the quantum supremacy frontier with a 281 Pflop/s simulation. *Quantum Science and Technology* 5, 3 (2020), 034003. DOI: https://doi.org/10.1088/2058-9565/ab7eeb

[189] Trevor Vincent, Lee J. O'Riordan, Mikhail Andrenkov, Jack Brown, Nathan Killoran, Haoyu Qi, and Ish Dhand. 2022. Jet: Fast quantum circuit simulations with parallel task-based tensor-network contraction. *Quantum* 6 (2022), 709. DOI: https://doi.org/10.22331/q-2022-05-09-709

[190] Hanrui Wang, Yongshan Ding, Jiaqi Gu, Yujun Lin, David Z. Pan, Frederic T. Chong, and Song Han. 2022. Quantumnas: Noise-adaptive search for robust quantum circuits. In *Proceedings of the 2022 IEEE International Symposium on High-Performance Computer Architecture*. IEEE, 692–708.

[191] G. Wendin. 2017. Quantum information processing with superconducting circuits: A review. *Reports on Progress in Physics* 80, 10 (2017), 106001. DOI: https://doi.org/10.1088/1361-6633/aa7e1a

[192] M. Werninghaus, D. J. Egger, and S. Filipp. 2021. High-speed calibration and characterization of superconducting quantum processors without qubit reset. *PRX Quantum* 2, 2 (2021), 020324. DOI: https://doi.org/10.1103/PRXQuantum.2.020324

[193] Steven R. White. 1992. Density matrix formulation for quantum renormalization groups. *Physical Review Letters* 69, 19 (1992), 2863.

[194] Dennis Willsch, Madita Willsch, Fengping Jin, Kristel Michielsen, and Hans De Raedt. 2022. GPU-accelerated simulations of quantum annealing and the quantum approximate optimization algorithm. *Computer Physics Communications* 278 (2022), 108411. DOI: https://doi.org/10.1016/j.cpc.2022.108411

[195] Karen Wintersperger, Florian Dommert, Thomas Ehmer, Andrey Hoursanov, Johannes Klepsch, Wolfgang Mauerer, Georg Reuber, Thomas Strohm, Ming Yin, and Sebastian Luber. 2023. Neutral atom quantum computing hardware: Performance and end-user perspective. *EPJ Quantum Technology* 10, 1 (2023), 32.

[196] Nicolas Wittler, Federico Roy, Kevin Pack, Max Werninghaus, Anurag Saha Roy, Daniel J. Egger, Stefan Filipp, Frank K. Wilhelm, and Shai Machnes. 2021. Integrated tool set for control, calibration, and characterization of quantum devices applied to superconducting qubits. *Physical Review Applied* 15, 3 (2021), 034080. DOI: https://doi.org/10.1103/PhysRevApplied.15.034080

[197] Anbang Wu, Gushu Li, Hezi Zhang, Gian Giacomo Guerreschi, Yufei Ding, and Yuan Xie. 2022. A synthesis framework for stitching surface code with superconducting quantum devices. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. Association for Computing Machinery, New York, NY, USA, 337–350. DOI: https://doi.org/10.1145/3470496.3527381

[198] Xin-Chuan Wu, Sheng Di, Emma Maitreyee Dasgupta, Franck Cappello, Hal Finkel, Yuri Alexeev, and Frederic T. Chong. 2019. Full-state quantum circuit simulation by using data compression. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM. DOI: https://doi.org/10.1145/3295500.3356155

[199] Xiaosi Xu, Simon Benjamin, Jinzhao Sun, Xiao Yuan, and Pan Zhang. 2023. A herculean task: Classical simulation of quantum computers. (2023). arXiv:2302.08880. Retrieved from https://arxiv.org/abs/2302.08880

[200] Ge Yan, Wenjie Wu, Yuheng Chen, Kaisen Pan, Xudong Lu, Zixiang Zhou, Yuhan Wang, Ruocheng Wang, and Junchi Yan. 2024. Quantum Circuit Synthesis and Compilation Optimization: Overview and Prospects. (2024). arXiv:2407.00736. Retrieved from https://arxiv.org/abs/2407.00736

[201] C. Young, A. Safari, P. Huft, J. Zhang, E. Oh, Ravikumar Chinnarasu, and M. Saffman. 2022. An architecture for quantum networking of neutral atom processors. *Applied Physics B* 128 (2022). DOI: https://doi.org/10.1007/s00340-022-07865-0

[202] Kieran Young, Marcus Scese, and Ali Ebnenasir. 2023. Simulating Quantum Computations on Classical Machines: A Survey. (2023). arXiv:2311.16505. Retrieved from https://arxiv.org/abs/2311.16505

[203] Boyuan Zhang, Bo Fang, Qiang Guan, Ang Li, and Dingwen Tao. 2023. HQ-Sim: High-performance state vector simulation of quantum circuits on heterogeneous HPC systems. In *Proceedings of the 2023 International Workshop on Quantum Classical Cooperative*. Association for Computing Machinery, New York, NY, USA, 1–4. DOI: https://doi.org/10.1145/3588983.3596679

[204] Chi Zhang, Ari B. Hayes, Longfei Qiu, Yuwei Jin, Yanhao Chen, and Eddy Z. Zhang. 2021. Time-optimal qubit mapping. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. Association for Computing Machinery, New York, NY, USA, 360–374. DOI: https://doi.org/10.1145/3445814.3446706

[205] Chen Zhang, Zeyu Song, Haojie Wang, Kaiyuan Rong, and Jidong Zhai. 2021. HyQuas: Hybrid partitioner based quantum circuit simulation system on GPU. In *Proceedings of the ACM International Conference on Supercomputing*. Association for Computing Machinery, New York, NY, USA, 443–454. DOI: https://doi.org/10.1145/3447818.3460357

[206] Chaoyu Zhang, Hexuan Yu, Yuchen Zhou, and Hai Jiang. 2021. High-performance and energy-efficient fpga-gpu-cpu heterogeneous system implementation. In *Advances in Parallel and Distributed Processing, and Applications: Proceedings from PDPTA'20, CSC'20, MSV'20, and GCC'20*. Springer, 477–492.

[207] Xin Zhang, YaQian Zhao, RenGang Li, XueLei Li, ZhenHua Guo, XiaoMin Zhu, and Gang Dong. 2019. The Quantum Shor Algorithm Simulated on FPGA. In *Proceedings of the 2019 IEEE Intl Conf on Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking*. 542–546. DOI: https://doi.org/10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00082

[208] Yilun Zhao, Yanan Guo, Yuan Yao, Amanda Dumi, Devin M. Mulvey, Shiv Upadhyay, Youtao Zhang, Kenneth D. Jordan, Jun Yang, and Xulong Tang. 2022. Q-GPU: A recipe of optimizations for quantum circuit simulation using GPUs. In *Proceedings of the 2022 IEEE International Symposium on High-Performance Computer Architecture*. 726–740. DOI: https://doi.org/10.1109/HPCA53966.2022.00059

[209] Ya-Qian Zhao, Ren-Gang Li, Jin-Zhe Jiang, Chen Li, Hong-Zhen Li, En-Dong Wang, Wei-Feng Gong, Xin Zhang, and Zhi-Qiang Wei. 2021. Simulation of quantum computing on classical supercomputers with tensor-network edge cutting. *Physical Review A* 104, 3 (2021), 032603.

[210] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, et al. 2020. Quantum computational advantage using photons. *Science* 370, 6523 (2020), 1460–1463. DOI: https://doi.org/10.1126/science.abe8770 arXiv:https://www.science.org/doi/pdf/10.1126/science.abe8770

[211] Henry Zou, Matthew Treinish, Kevin Hartman, Alexander Ivrii, and Jake Lishman. 2024. LightSABRE: A Lightweight and Enhanced SABRE Algorithm. (2024). arXiv:2409.08368. Retrieved from https://arxiv.org/abs/2409.08368

[212] Kasia Świrydowicz, Noel Chalmers, Ali Karakus, and Tim Warburton. 2019. Acceleration of tensor-product operations for high-order finite element methods. *The International Journal of High Performance Computing Applications* 33, 4 (2019), 735–757. DOI: https://doi.org/10.1177/1094342018816368 arXiv:https://doi.org/10.1177/1094342018816368

Received 14 October 2024; revised 10 June 2025; accepted 3 August 2025