ISSN 2224-087X. Електроніка та інформаційні технології. 2023. Випуск 21. С. 90–107
Electronics and information technologies. 2023. Issue 21. P. 90–107

UDC 004.4 DOI: https://doi.org/10.30970/eli.21.9

OPEN DIRECTIONS IN QUANTUM SOFTWARE STACK: FROM NISQ to QUANTUM UTILITY

M. Tsymbalista, M. Maksymenko, I. Katernyak

Ivan Franko National University of Lviv,
General Tarnavsky Street, 107, 79017 Lviv, Ukraine

ihor.katernyak@lnu.edu.ua

Improvement in Quantum Computing (QC) performance will allow us to solve a wide range of complex problems that classical computers of today can't handle. Nowadays we feel closer to achieving a state of Quantum Utility (QU) than ever before. Importance for both academia and business to understand the current state of technology and tooling, their extensibility points along with perspective optimization algorithms are on the verge. The purpose of the article is to provide a structured analysis of existing progress in QC. It serves a purpose of a guide on where significant progress is expected in the upcoming years, outlines details about tooling to start doing experiments, and introduces Quantum Computing Optimisation Middleware (QCOM) reference architecture as a backbone around which new building blocks are expected to be added in the upcoming years e.g. industry-specific enterprise connectors. The article reviews only the software stack, not considering opportunities for hardware as they seem to be much further away. This should help scientists and engineers to define a mental model of how to move forward to reach both mid and long-term goals toward QU.

Keywords: quantum computing, quantum utility, quantum algorithm performance, Quantum Computing Optimisation Middleware, error correction, qubit mapping.

Abbreviations

QC - Quantum Computing
QS - Quantum Supremacy
QA - Quantum Algorithm
QU - Quantum Utility
QH - Quantum Hardware
ML - Machine Learning
RC - Random Circuit
PF - Programming Framework
API - Application Programming Interface
SDK - Software Development Kit
QCOM - Quantum Computing Optimisation Middleware
QPU - Quantum Processing Unit
NISQ - Noisy Intermediate Scale Quantum
HPC - High-Performance Computing
IR - Intermediate Representation
SDK - Software Development Kit

Overview

QC domain which includes hardware, software, tools, algorithms, etc. is usually perceived as more complex in comparison to classical technology. The knowledge base for the domain is limited and is mostly driven by research studies and technical documents of a few tools that were created by leaders of the industry. The next breakthrough in QC is going to happen at the intersection of algorithms (not high-level algorithms for solving specific problems from let's say physics domain, but algorithms on the compilation level of quantum computing stack) and existing tools (programming languages, software libraries or compilers). These programming languages and compilers have knowledge instilled of how to manipulate physical qubits on specific hardware, translate them to virtual ones, perform error correction and expose a high-level interface that hides most of the complexity to allow efficient implementation of QAs.

There is a number of publications around the basics of quantum computing and tools [1], publications that get under the hood of what is happening during compilation [2], studies of the performance of different compilers along with details of the tool that allow the combination of compilation steps from different vendors [3]. Also, studies that show optimization algorithms for circuit mapping [20], improvement of error correction [16], etc.

Despite all that, it is hard to make a conclusion on how to contribute to progress on promising directions and approaches to achieve Quantum Utility (QU), a term coined by in [29] to characterise practicality of modern quantum processing units (QPUs). Details of technological interfaces in tools around which performance optimizations could be tried are not summarized. The same for compilation layer phases with details of potential optimization benefits along with nascent algorithms of each phase that could contribute to breakthrough.

The purpose of the study is to analyze the current state of the industry and outline the most recent challenges and opportunities ascending in the field from the software perspective. This should help researchers and software developers to get a better feeling about potential ways of moving closer to what is called QU.

Methodology

Paper utilizes methods from qualitative research. Analysis of the most recent studies that have been conducted over the course of three years, to draw a picture of existing QC eco-system state and potential points of breakthrough. Also, an interview has been conducted with the co-founder of Haiqu [33] - Mykola Maksymenko [34] who helped to distil the course of thinking about the most recent challenges in QC, and provided guidance on tooling along with approaches that could be used to measure the performance of experiments in the space.

From Quantum Supremacy to Quantum Advantage and Utility.

These concepts are similar, but the difference is important to grasp. In 2019 Google claimed that they've achieved QS [4]. Even though calculation had no practical value it layed the beginning of what is called QS - the ability to solve a problem (even with no practical value) in a reasonable time frame, in contrast to classical computer architecture that would take infinity to produce a result. In 2023 IBM released a research [6] on simulating the 127-spin random Ising model which has a near-term practical potential in providing new physical insights to non-equilibrium dynamics of the underlying quantum state.

Quantum advantage is a very close neighbor of QS drawing a separation between "not practical" and "measured practical success" or "breakthrough innovation" over the most powerful computer in a particular domain e.g. physics, or drug discovery. While being far from providing practical utility recent experiments provide insights and outlook for near-term developments in NISQ era.

Measuring improvement in performance in quantum compilation

As NISQ devices are prone to errors and imperfections it is vital to optimize near-term Quantum Algorithms for performance by minimizing the number of operations and adjusting algorithm execution to a particular hardware specification. To this end, Quantum Compilation tools are already a vital component of a near-term software stack.

Quantum compilation encompasses phases like converting a QA into gate instructions, mapping and routing qubits on architectures with connectivity constraints, optimizing circuits, translating gates into a hardware's native gate set, and scheduling quantum operations on physical hardware. Among these, circuit optimization plays a crucial role in the compilation process. There are several baseline metrics that are a cornerstone for measuring the performance of algorithms. The first set of metrics is the Compression Ratio (CR) for the number of gates and gate depth:

$$CR(\,gate\ count, g \in G) = \frac{gate\ count\left(stage_{in}\right)}{gate\ count\left(stage_{out}\right)}\ ,$$
$$CR(\,gate\ depth, g \in G) = \frac{gate\ depth\left(stage_{in}\right)}{gate\ depth\left(stage_{out}\right)}\ . \tag{1}$$

CR (1) measures a decrease in circuit size before and after circuit optimization. Only a specific gate class is considered (G); g is a gate that belongs to G gate class; *gate count* refers to the number of individual quantum gates required to execute a specific QA or circuit; *gate dept* refers to the total number of sequential layers of quantum gates required to implement a specific algorithm or circuit; $stage_{in}$, $stage_{out}$ input and output circles respectively. A gate layer is a collection of gates that can be executed in parallel, meaning all gates within a layer are applied simultaneously.

One of the most important metrics is Circuit Cost Function (C):

$$C = -D\log K - \sum_i \log F_i^{1q} - \sum_j \log F_j^{2q}\ , \tag{2}$$

where C - circuit cost, D - circuit depth, K - a factor that penalizes deep circuits, F_i^{1q} - fidelity of single-qubit gates, F_j^{2q} - fidelity of two-qubit gates. Fidelity quantifies how closely the output state of the gate matches the expected or ideal state. The expression inside log in formula (2) might be understood as an aggregate fidelity of the complete circuit, presuming that the fidelity of a series of quantum gates can be expressed as the product of the individual gate fidelities. Details about the fidelities of particular gates could be taken from public resources shared by vendors. The parameter K penalizes circuits with higher depth. It is physical interpretation tied to the practical challenges and limitations of implementing longer quantum circuits, where higher gate depths can lead to increased susceptibility to errors. This formula is quite simplified and in case there is interest to explore more details of what else could impact the cost model, please refer to study [3] which provides more details. Also, there are ongoing studies on how to create neural-network-based approximators that predict hardware-specific noise levels for a quantum circuit [21]. Summarized values for gate fidelities and parameter *K* could be found in [3].

Circuit cost function improvement (ratio) is defined as:

$$Cost\ improvement = \frac{Cost(stage_{in})}{Cost(stage_{out})} \quad , \tag{3}$$

where proportion divides Circuit Cost Function for input and output circles respectively.

Another metric is related to the concept of random circuits. This concept refers to sequences of quantum gates that are applied to qubits in a random or pseudo-random manner. It was introduced for research purposes because the most promising algorithms are out of reach in today's NISQ era. RC is defined by gate sets and probabilities assigned to gate type. So the metric is named: a quantitative measure of the density of a particular gate class *G* and is notated as (*p*)

$$p(g \in G) = \frac{gc(G)}{gc_{total}} \quad , \tag{4}$$

where gc is gate count for the gates of class *G*, gc_{total} – total gate count. The effectiveness of circuit compression is closely linked to the selection of the gate set and the density of two-qubit gates within random circuits. Circuit Equivalence Verification aims to determine if two quantum circuits perform the same quantum computation, producing identical outcomes for all possible input states. It is very important in the field of optimizations. It is defined by calculating classical fidelity between two probability distributions:

$$\mathcal{F}_{cl} = \sum_{\sigma \in \{0,1\}^N} \sqrt{p_{in}(\sigma)\, p_{out}(\sigma)} \quad , \tag{5}$$

where $p_{in}(\sigma) p_{out}(\sigma) = |\psi_{in/out}(\sigma)|^2$ are the probability distributions of measured bitstrings (σ), $\psi_{in/out}(\sigma)$ - denote the 2^N dimensional state vector of optimized *N*-qubit circuits. Summation in (5) is performed over all bitstrings σ of length *N*, that enumerate components of the state vector. A bitstring is a sequence of binary digits, where each digit is a "bit" representing a binary value of either 0 or 1. Despite classical, it is also relevant to QC.

Study [3] relies heavily on the metrics outlined above. They automated them in their Automated Benchmarking Platform for Quantum Compilers. It is strongly encouraged to learn how to use the tool and get a feel of how different compilers perform distinct stages. It could serve as a baseline verification tool when brainstorming ideas and researching new optimization algorithms. Authors also introduce a very interesting approach of combining subroutines from different modules in a specific way in order to achieve better results, the extension of this idea is presented in a recent work by Quetschlich [30, 31]. It should also be a must-do thing to investigate when analyzing the platform and approach.

How to approach initial attempts of QC performance improvement. Extensibility of compilation pipelines.

In recent years, the progress in QC has led to the emergence of various tools for quantum development, encompassing standalone simulators, SDKs, programming languages, etc. A very nice intro and comparison of capabilities are provided in the study [1]. Given the differences in their approaches, target applications, and underlying platforms, understanding the nature of these tools is essential for harnessing the computational potential of QAs efficiently and seamlessly. Most of them are used for application-level algorithms implementation using a pre-defined set of compilation phases that handle the whole compilation process and execution on real hardware.

In order to experiment with performance improvement of algorithms we need to get under the hood of compilation and customize different steps. As it is in every opinionated framework or tool, those choices are limited, but they significantly reduce overhead and allow to implement hypotheses quickly without a need to build a new compiler from scratch. From the other perspective it not going to work for more complex experiments. It is vital to provide a quick overview of some popular tools.

Qiskit Terra. Transpiler provides extensibility points with its Transpiler Passes and Pass Manager [14]. The package allow us to write new transpiler passes (circuit transformations) and combine them with different existing passes, handling their order. The whole pipeline operates under the orchestrator (PassManager). It is responsible for communication between stages (passes) and their scheduling. It is worth mentioning various optimization levels provided through pre-defined pass managers that reflect additional complexity of compilation pipeline. There are optimization levels 0 to 3. The higher the number, the more optimized circuit will be generated. This number depends very much on actual hardware and of course on the algorithm.

- Level 0: Simply associate the circuit with the backend, without explicit optimization apart from any optimizations performed by the mapper.
- Level 1: Performs circuit mapping and additionally conducts minor optimizations by combining neighboring gates.
- Level 2: Moderate optimization is employed, incorporating a noise-responsive arrangement and a gate-cancellation process relying on gate commutation correlations.
- Level 3: Extensive optimization is implemented, encompassing previous stages and including the resynthesis of two-qubit gate blocks within the circuit.

There is a number of frameworks and tools that extend capabilities and quality of Qiskit compilation tools. For example, Fire Opal (one of the leading commercial product in quantum optimizations space) employs Qiskit's preset pass manager with optimization level 3 as its foundation and showcases consistent improvement of the result vs out-of-the-box optimization.

Qiskit Pulse API [22]. It is a pulse-level quantum SDK. This level allows more control when programming QH. Achieving optimal QH performance necessitates instantaneous pulse-level instructions. Pulse delivers precisely that, empowering scientists to define experiment dynamics with precise timing. This SDK is particularly influential in enhancing error mitigation methodologies. A very insightful study of how Pulse API can be leveraged is provided in reference [19]. In this study, they utilized the Pulse API to practically evaluate numerically optimized error-robust pulses for implementing single-qubit gates on IBM's cloud-based quantum computers. The general concept of how API works is that arbitrary time-ordered signals (instructions) are provided as input, concurrently scheduled across multiple virtual hardware or simulator resources (channels). The system also facilitates the user in reconstructing the time dynamics of the measured output.

Q# compiler extensions [23]. For those, closer to the Microsoft eco-system, custom compilation steps allow us to extend and customize the Q# compilation process similar to the Qiskit Transpiler approach. Similar to any compiler, the Q# compiler follows a sequence of compilation stages. The initial steps involve parsing and validating the Q# code, and generating a data structure that captures the compilation's state. Subsequent stages traverse this data structure to generate updated versions. These stages often involve tasks like simple optimizations, constant folding, loop unrolling, as well as function and operation inlining. With custom compilation steps there is a possibility to get into the pipeline of execution, so custom, more complex optimizations could be done.

This is not an extensive list of the tools with plugin points, but they are sufficient to start trying custom optimization approaches.

Quantum Computing Optimisation Middleware

Most of the studies today look at the QC process from the perspective of layered architecture which has been inspired by the OSI networking model [10]. Since 2012 things have changed in the field and today reasoning about QC algorithm solely from this perspective doesn’t provide detailed visibility of internals.

Q-CTRL pioneered the term “Quantum infrastructure software” [11] which they sell as a complex solution comparable to VMware, Citrix, and similar complex virtualization technologies in classical computing. These systems are heavier. The purposes of those systems are different for quantum and classical computing. Virtualization is a layer over the operating system in cases where a hypervisor is used. In QC this layer is not present, so middleware seems to be a better word to reason about the concept. QCOM (Fig. 1) is a better term to describe a software layer that is responsible for the performance optimization of arbitrary QAs. It shouldn’t serve the purpose of doing commodity compilation/transpilation which is done via proxy QC PF, but focus on enriching its capability, serving as a vitamin for researchers of QAs, to allow them to solve real-world problems with QC. It should be able to do optimization on both logical-level circuits and hardware circuits. The circuit processing in QC can basically be split into logical (higher-level theoretical computation) and physical (lower-level physical implementation) levels. A circuit at the logical level is defined using abstract, discrete-time gates and classical computations executed in real time. This is transformed by a compiler into a circuit at the physical level, where quantum operations are realized using continuous, time-varying signals adhering to specific timing constraints. There is still an uncertain question of whether proprietary compilers could allow for sufficient flexibility in the long run, so potentially QCOM could be responsible for compilation until both product segments will mature and have better, more standardized interfaces between each other.

QCOM could be exposed as a package plugin for different QC libraries, which calls a complex optimization layer running in the cloud. This software layer will be driving the industry in the upcoming decade. It will be called one way or another, but there will be dozens of companies working on it in some form, leveraging techniques from HPC, ML, and more classical QAs until the point of reaching QU where this component could grow into something else like let’s say native to QH firmware.

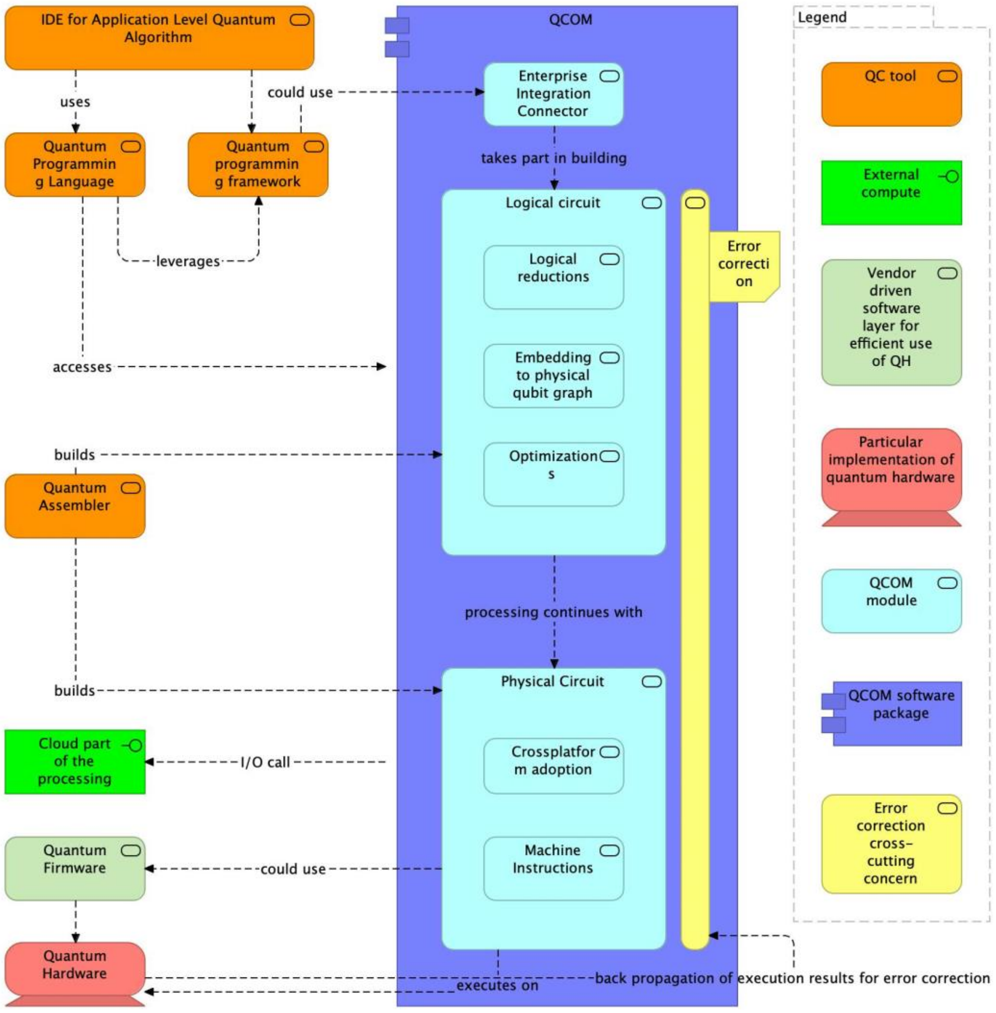
IDE for Application Level Quantum Algorithm
uses
could use
Quantum Programming Language
Quantum programming framework
leverages
accesses
builds
Quantum Assembler
builds
Cloud part of the processing
I/O call
Quantum Firmware
could use
Quantum Hardware
executes on
back propagation of execution results for error correction
QCOM
Enterprise Integration Connector
takes part in building
Logical circuit
Logical reductions
Embedding to physical qubit graph
Optimizations
processing continues with
Physical Circuit
Crossplatform adoption
Machine Instructions
Error correction
Legend
QC tool
External compute
Vendor driven software layer for efficient use of QH
Particular implementation of quantum hardware
QCOM module
QCOM software package
Error correction cross-cutting concern

Fig 1. Conceptual architecture of QCOM and its interactions with different QC tools, external components, and hardware.

Logical reductions. Refer to techniques or processes used to simplify or transform a quantum circuit or logical expression into a more concise or manageable form while preserving its computational equivalence or logic. These reductions aim to make quantum computations more efficient, understandable, or amenable to analysis. They involve identifying patterns, redundancies, or equivalent operations within the quantum circuit or algorithm and simplifying them without changing the final outcome. Logical reductions can encompass various techniques, such as gate fusion, gate commutation optimization, constant propagation, gate cancellation, simplification of logical expressions, etc.

Embedding to physical qubit graph. Embedding to a physical qubit graph often referred to as qubit mapping or qubit allocation, is a crucial step in QC when running QAs on real quantum processors. The physical qubit graph represents the connectivity and layout of qubits on a specific quantum device. In quantum computers, qubits are not always fully connected to each other due to limitations in hardware design and qubit connectivity. This means that qubits can only directly interact with certain neighboring qubits, and operations involving non-adjacent qubits need to be decomposed into sequences of single-qubit and two-qubit gates.

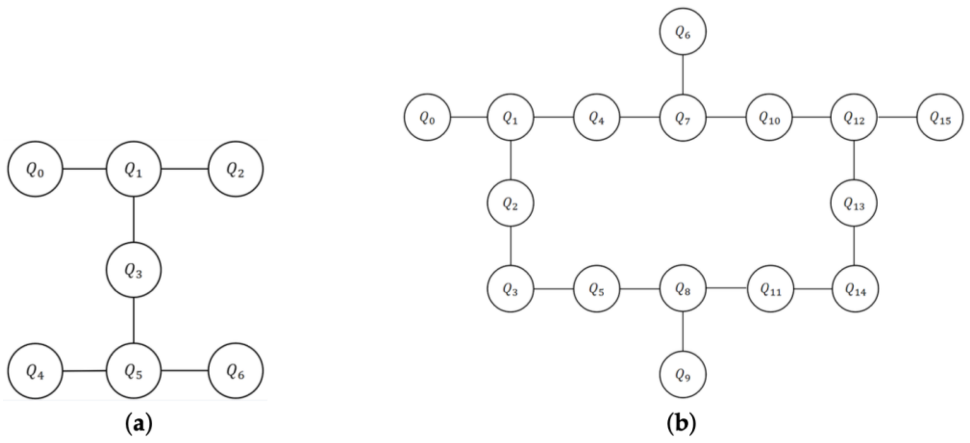

Fig 2. Quantum topology. (a) Ibm_perth. (b) Ibmq_guadalupe.

Given the restricted interconnections among physical qubits in current quantum devices as outlined on Fig. 2, identifying an initial quantum graph that accommodates all double qubit gates across the entire circuit is commonly considered challenging. Consequently, the requirement for mapping algorithms and optimization approaches arises to overcome this limitation and identify the optimal arrangement of physical qubits. This process might necessitate the introduction of extra qubit gates to offset the deficiency in connectivity. Fig. 3 (a, b) outlines how circuit could be modified in the way that outcome representation is equivalent. Quantum circuits could be represented with a gate dependency graph as outlined in Fig. 3 (c).

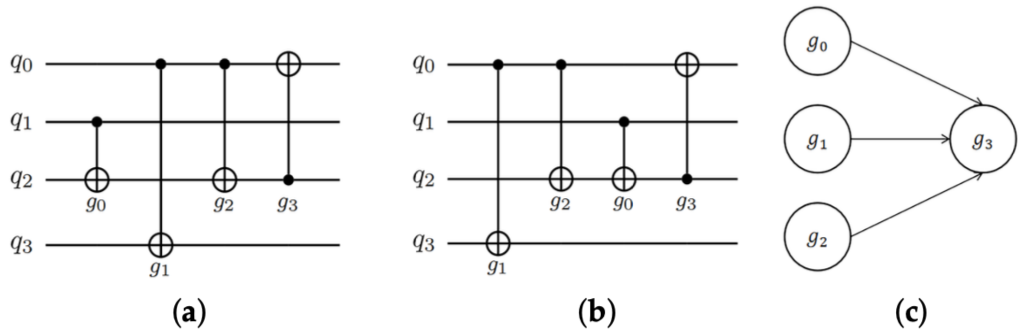

Fig. 3. Different equivalent representations of quantum circuits. (a) Example of a quantum circuit. (b).

The circuit is modified by interchanging the order of commuting gates. (c) Gate dependency graph.

Here's how the embedding process works:

- Logical Qubits: The QA is formulated in terms of logical qubits, which represent the quantum information being processed.
- Physical Qubits: The quantum device has a specific layout of qubits with certain connectivity constraints. The embedding process involves finding a mapping between logical qubits and available physical qubits on the device.
- Connectivity Constraints: The mapping needs to respect the connectivity constraints of the device. If two logical qubits interact in the algorithm, but they are not directly connected to the device, additional gates need to be added to create an equivalent circuit that can be executed on the hardware.
- Gate Decomposition: If the algorithm involves gates between non-adjacent qubits, these gates need to be decomposed into a series of single-qubit and two-qubit gates that can be executed on the available physical qubits.
- Optimization: The embedding process often involves optimization techniques to minimize the additional gates, preserve the QA's logical behavior, and mitigate the impact of noise and errors.
- Performance Considerations: The quality of the embedding can impact the overall performance of the QA, affecting factors such as gate fidelity, execution time, and error rates.

There are a lot of opportunities at the current level. For example, the paper [20] uses the circuit template matching optimization method which accounts for connectivity constraints of different topologies. Method satisfies those constraints by cutting the number of gates. The impressive thing about it is that it outperforms Qiskit across not only one but various IBM hardware architectures. It could serve as a baseline for further research in the direction that could incorporate more complex algorithms. Another family of approaches relies heavily on ML. Authors in the study [30] prove that it is possible for ML systems to learn based on historical data as there are many circuits executed on QH every day. These techniques are closer to cloud vendors which could instill that as part of their policy. Study [20] leverages a deep reinforcement learning approach that doesn’t rely on batches of historical data but could learn along the way. Reinforcement learning techniques strive to acquire an action policy that dictates the appropriate action to take based on specific observations of the current state. The goal is to maximize a cumulative reward function. Within the proposed framework, observations are derived by extracting specific attributes from the quantum circuits at each state. Furthermore, a sparsely defined reward function is employed to indicate the achievement of a final state and subsequently assess the "quality" of the resulting circuit. This assessment could pertain to factors such as the resulting gate count, circuit depth, or anticipated fidelity.

Paper [9] concentrates on the problem of qubit routing leveraging a new decomposition approach based on the capabilities provided by integer programming, which also shows positive intermediate results. Qubit mapping focuses on the logical-to-physical qubit assignment, ensuring that the quantum circuit's logical qubits are placed on available physical qubits. Qubit routing, on the other hand, involves determining the pathways that qubit states will take as they move through the hardware during gate operations.

Embedding to a physical qubit graph is a critical challenge in QC, especially for near-term devices with limited qubit connectivity. Effective embedding strategies are necessary to successfully run QAs on real hardware and bridge the gap between theoretical algorithms and practical execution on quantum processors. Research in that direction has been booming over the last couple of years with a lot of studies that use different approaches, so it is expected that its going to have a near-term impact on reaching the state of QU.

Optimizations. This is a layer that is responsible for the optimization and simplification of quantum circuits, particularly for near-term quantum devices with limited resources and high error rates. Some of the techniques that could be used:

- N-Qubit Blocks Clustering. This refers to the grouping or clustering of multiple adjacent or non-adjacent qubits that interact together in a quantum circuit. Clustering qubits that frequently interact in the circuit can reduce the need for repeated qubit swaps or additional gates, which can help mitigate errors and improve circuit performance. By optimizing the arrangement of qubits based on their interaction patterns, joint n-qubit blocks clustering aims to enhance the efficiency of quantum computations.
- 1-Qubit Optimization. This involves optimizing and simplifying the operations applied to individual qubits (1-qubit gates) within a quantum circuit. By minimizing the number of gates or finding gate sequences that are more robust against errors, 1-qubit optimization aims to improve the overall quality of the circuit. Effective 1-qubit optimization techniques can lead to more reliable quantum computations, especially on noisy devices.
- Blocks Consolidation. Refers to the process of identifying sequences of gates that can be combined or condensed into more efficient operations. By identifying patterns or sequences of gates that can be optimized, blocks consolidation reduces the overall gate count and complexity of the circuit. This can result in faster execution times and reduced susceptibility to errors.

These concepts collectively contribute to making quantum circuits more suitable for execution on near-term quantum hardware. They address challenges posed by limited qubit connectivity and noise. The goal is to create more compact, efficient circuits that can be executed with higher fidelity on currently available quantum processors.

Error correction cross-cutting concern. In traditional software architecture, the cross-cutting concern is referred to a specific aspect of software that spans several logical layers of an application (logging, authentication, .etc). In QC error correction techniques are applied on every layer of program compilation and even as part of post-processing after QA execution, so it is a good candidate to be considered as one of the most important cross-cutting concerns of quantum software architecture.

Noise and errors in QH are one of the core challenges that arise due to the delicate nature of quantum systems and the influence of their surrounding environment. Before diving into the details of how they could be handled it is important to understand why they are happening in the first place. The main causes are:

- Decoherence*:* Occurs when qubits, sensitive to environmental interactions like temperature fluctuations, lose their delicate quantum states and coherence, resulting in mixed quantum information and computational errors.
- Quantum gate imperfections: Arise from challenges in precisely implementing quantum gates due to factors like imperfect qubit interactions, fluctuating control parameters, and variations in qubit properties, leading to errors in quantum operations.
- Crosstalk in quantum processors: occurs when closely positioned qubits interact, resulting in an unintended coupling that disrupts desired individual qubit operations and introduces errors.
- Gate duration and speed: are crucial to prevent error accumulation; while executing gates too quickly can introduce errors from imperfect control and noise, implementing them too slowly can also lead to errors due to increased susceptibility to decoherence during the operation.
- Readout errors: stem from imperfections in measurement devices, signal amplification, and hardware components, resulting in inaccuracies when determining the final state of a qubit.
- Error Correction Overhead: Necessitates extra qubits and operations for encoding, detecting, and correcting errors, which in turn can introduce noise and errors, potentially impacting computational efficiency.
- Qubit Imperfections: arise from manufacturing processes and material flaws, resulting in variations in properties like energy levels, transition frequencies, and coupling strengths.

Considering that there are a lot of factors that cause errors it is worth summarising what strategies could be considered to deal with them:

1. Error Suppression. Techniques aimed to prevent errors from happening. They reduce the likelihood of hardware error while quantum bits are being manipulated or used for memory storage. It leverages the nature of quantum control. More details could be found in a study [16]. Also, ML could help to increase the robustness of quantum gates [17]. Other strategies are described in [18] [19]. Most of the studies are done by the Q-CTRL team. The assumption is that they use those approaches as part of their commercial product FireOpal. This strategy is not effective for problems like “Energy Relaxation”.
2. Error Mitigation. Errors could happen during algorithms execution and also while measuring output. Various approaches have been devised to address them, aiming to enhance outcomes through postprocessing. These strategies encompass diverse methods such as randomized compiling, measurement-error mitigation, zero-noise extrapolation, and probabilistic error cancellation, yet they exhibit shared implementation principles. In general, error mitigation strategies entail running numerous slightly varied iterations of a target algorithm and subsequently combining outcomes. The adjustments made to the circuit can either be randomized or follow a predetermined algorithm. Some form of this is used in a leading product of Q-CTRL -

FireOpal. But due to the high costs of running algorithms several times, for the time being, it is a less effective strategy in comparison to Error Suppression strategies.

3. Quantum Error Correction. This strategy entails the creation of algorithms that are aimed to identify and fix errors. In general, they work by implementing redundancy, distributing the state of qubits to other qubits. Then by checking helper qubits, it is identified whether an error happened or not. If yes, the correction could be applied. A huge disadvantage is the number of qubits that are required. As we all know, the amount of qubits is very limited in today's hardware. So as of today, this is not a very effective technique until better approaches are found. This is a huge opportunity for "rockstar" researchers. More insights on the approach could be found in [15].

Thorough examinations have revealed that quantum error suppression currently offers the most compelling demonstrated advantages and optimal adaptability for integration with error mitigation and quantum error correction, culminating in outcomes that exceed the cumulative impact of individual components. Details on the success of their combination could be found in [18].

Cross-platform adoption. Layer that is responsible for building an abstraction over different hardware implementations. It is more of an enterprise, nice to have feature, so we are not covering it in the scope of this analysis.

Machine instructions. The translation from quantum gates to pulse-level instructions is a complex and crucial step in making QAs executable on real hardware. It requires a deep understanding of the physical properties of the qubits, noise sources, and control mechanisms. Incorporates the following phases:

- Gates to Pulses (Gate Decomposition). The process begins by translating abstract quantum gates used in QAs into specific sequences of physical gates that are available on the QH. These physical gates are then further translated into corresponding time-varying signals (pulses) that control the behavior of qubits during gate operations. Translating gates to pulses involves calibration and characterization processes to fine-tune the parameters of the pulses.
- Timing Resolution. Timing resolution involves determining the precise timing intervals for applying pulses to qubits during gate operations. High timing resolution ensures accurate execution of gates, reducing the likelihood of qubits losing their quantum states due to timing errors.
- Pulse Optimization. Pulse optimization focuses on finding the optimal pulse shapes, durations, and timings to achieve desired gate operations. Optimization techniques, often involving classical computations, are used to minimize errors and improve gate fidelity by shaping the pulses in ways that mitigate noise and imperfections in the hardware.
- Control Flow Optimization. Involves enhancing the sequence of quantum operations (quantum gates) within a quantum circuit to optimize its execution efficiency and mitigate errors. Quantum control flow refers to the ordering of quantum gates and the management of quantum states as they evolve through the circuit. Optimizing control flow in quantum computing aims to improve gate fidelity, reduce decoherence effects,

and enhance the overall performance of quantum algorithms. Gate fusion, gate reordering, optimal control sequences, etc. are some of the examples of techniques used on this level.

This integrated process ensures that QAs are translated into executable instructions that account for physical hardware constraints, timing precision, pulse optimization, and control flow considerations. The goal is to maximize the reliability and efficiency of quantum computations on real quantum processors.

As part of this chapter comprehensive overview has been provided of steps that should be considered when building custom QCOM for both commercial and scientific purposes. In the next chapter, we will talk more about tooling, that allows us to implement those fine-grained manipulations.

Tooling to get into the aggressive scientific research of QC optimization. The backbone for building QCOM.

Continuing the discussion of extensibility, none of the tools have good support for developing next-phase technology for QC. Despite some extensibility points, they don't offer enough control of QH and algorithms execution. Interaction only on the gates level abstraction doesn't provide the ability to manage lower-level control sequences with the purpose to do calibration, error mitigation, and characterization. Therefore, it becomes imperative to possess the capability to regulate the timing and establish links between quantum instructions and their corresponding pulse-level executions across different physical implementations or technologies used to create and manipulate qubits in QC. Timing features are especially important for the characterization of decoherence, crosstalk [24], dynamical decoupling [25], etc.

All those features are present in OpenQASM (Open Quantum Assembly Language) [26]. As a programming language, it is designed to serve the purpose of intermediate representation (IR). It is used by upper-tier compilers to interact with QH, enabling the depiction of an extensive array of quantum operations along with classical feed-forward flow control based on measurement results. It natively supports abstractions of logical and physical levels via specific semantics. Control flow instructions can be used to program repeat-until-success algorithms [27] and magic state distillation protocols [28]. There are also many other examples that show the potential of OpenQASM to get us closer to QU. Gate modifiers are another important mechanism that allows the creation of new gates based on existing ones. For example, modifier for inverting ads more readability which greatly raises optimization opportunities (it is hard to understand context when gates are decomposed).

Program execution flow (Fig. 4). Specific segments are classical, amenable to writing in classical programming languages for near-time execution. The quantum program generates a payload for execution on QPU. This data package encompasses expanded quantum circuits and external real-time classical functions. External real-time classical functions refer to computational tasks or functions that are executed using classical computing resources, outside of the QPU. OpenQASM serves as the language for defining the quantum circuits, encompassing interface calls to external classical functions. There might be higher-order elements within the circuit data, subject to optimization prior to generating OpenQASM. An OpenQASM compiler can modify and optimize all aspects of circuits described through the IR, including basis gates, qubit mapping, timing, pulses, and control flow. The final physical circuit along with external functions is then processed by a target code generator to produce binaries intended for the QPU. Almost every aspect of program compilation could be controlled in detail. So as of today, OpenQASM can be considered a core tool for hard-core research of QC optimization and as a backbone for QCOM implementation.

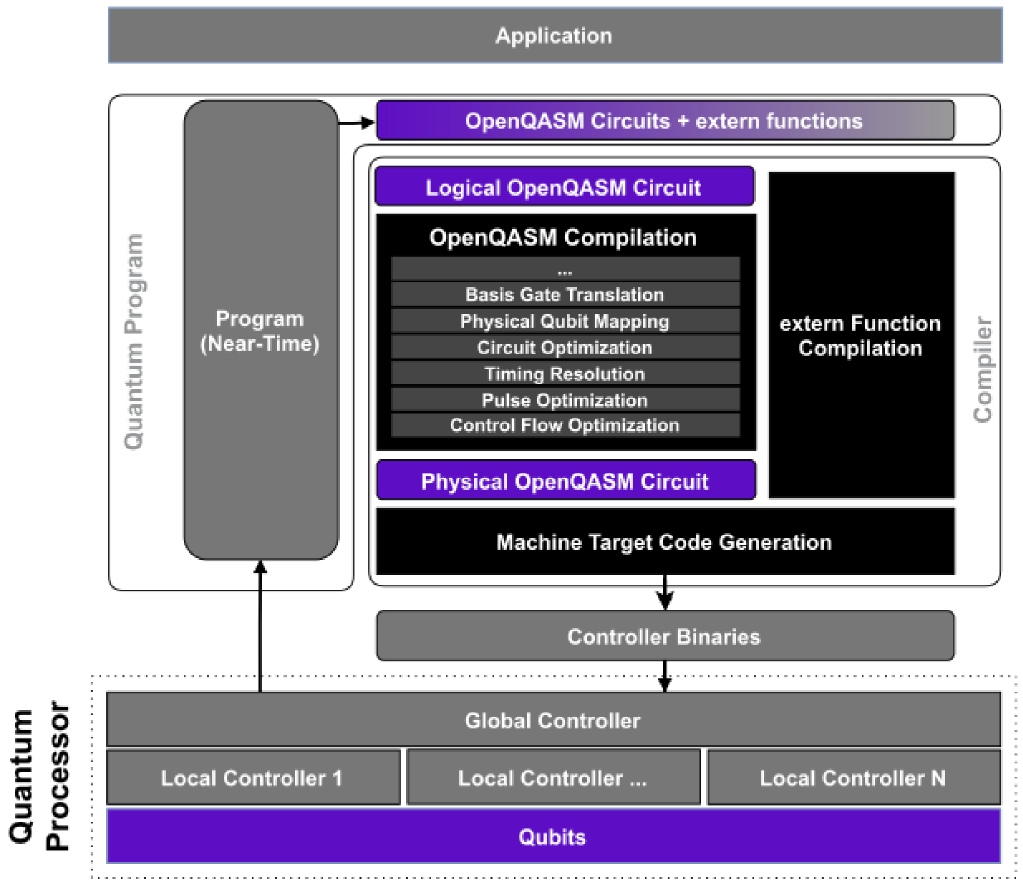

Fig. 4 [26]. The compilation and execution model of a quantum program, and OpenQASM's place in the flow.

Conclusion

This paper draws a picture of the current state of QC, looking into the most recent challenges and opportunities ascending in the field from the software perspective. QCOM conceptual architecture is proposed as an implementation backbone for optimization software spanning different layers of QA execution from trivial gate optimizations to more complex layout matching, manipulations on the machine-level instructions, and error correction. The motivation behind optimization on each layer is described along with interesting techniques that in combination increase the performance of the most popular tools like Qiskit. Analyses helped to identify priority directions for future research efforts, based on the recent progress in the industry, along with tools that could help to reach better control of quantum circuits execution. The proposed QCOM reference architecture aims to streamline the implementation of experiments and hypothesis testing in the field.

As the next steps in the research, prioritized directions that crystalized during this phase could be tackled: "Embedding to physical qubit graph" and "Error correction" (on both circuit and hardware layers). They showed a significant level of opportunity to reach QU in the nearest years. Covered optimization routines in different combinations could be implemented based on QCOM architecture, using OpenQASM programming language, and benchmarked using the Arline Benchmarks [32] tool to measure performance increase in comparison to well-established tools like Qiskit, etc. This will help to reason about how far we are from reaching QU and what could be done to get closer to it.

References.

[1] *Himanshu Sahu*, *Hari Prabhat Gupta*. Quantum Computing Toolkit from Nuts and Bolts to Sack of Tools. 2023

[2] *Marco Maronese*, *Lorenzo Moro*, *Lorenzo Rocutto*, *Enrico Prati*. Quantum Compiling. 2021

[3] *Y. Kharkov*, *A. Ivanova*, *E. Mikhantiev*, *A. Kotelnikov*. *Arline Benchmarks.* Automated Benchmarking Platform for Quantum Compilers. 2022

[4] Google Claims a Quantum Breakthrough That Could Change Computing. Retrieved from https://www.nytimes.com/2019/10/23/technology/quantum-computing-google.html

[5] What is quantum supremacy? Retrieved from https://www.techtarget.com/searchsecurity/definition/quantum-supremacy

[6] *Youngseok Kim*, *Andrew Eddins*, *Sajant Anand*, *Ken Xuan Wei*, *Ewout van den Berg*, *Sami Rosenblatt*, *Hasan Nayfeh*, *Yantao Wu*, *Michael Zaletel*, *Kristan Temme*, *Abhinav Kandala*. Evidence for the utility of quantum computing before fault tolerance. 2023

[7] Why quantum 'utility' should replace quantum advantage. Retrieved from https://techcrunch.com/2021/11/11/why-quantum-utility-should-replace-quantum-advantage/

[8] What Is NISQ Quantum Computing? Retrieved from https://thequantuminsider.com/2023/03/13/what-is-nisq-quantum-computing/

[9] *Friedrich Wagner*, *Andreas Bärmann*, *Frauke Liers*, *Markus Weissenbäck*. Improving Quantum Computation by Optimized Qubit Routing. 2023

[10] *N. Cody Jones*, *Rodney Van Meter*, *Austin G. Fowler*, *Peter L. McMahon*, *Jungsang Kim*, *Thaddeus D. Ladd*, *Yoshihisa Yamamoto*. Layered architecture for quantum computing. 2012

[11] Quantum infrastructure software. Retrieved from https://q-ctrl.com/topics/quantum-infrastructure-software

[12] Differentiating quantum error correction, suppression, and mitigation. Retrieved from https://q-ctrl.com/topics/differentiating-quantum-error-correction-suppression-and-mitigation

[13] *A. Paler*, *L. M. Sasu*, *A.-C. Florea*, *R. Andonie*. Machine learning optimization of quantum circuit layouts, ACM Transactions on Quantum Computing. 2022

[14] Transpiler Passes and Pass Manager. Retrieved from https://qiskit.org/documentation/tutorials/circuits_advanced/04_transpiler_passes_and_passmanager.html

[15] Quantum error correction. Retrieved from https://q-ctrl.com/topics/quantum-error-correction

[16] *Harrison Ball, Michael J. Biercuk, Andre Carvalho, Jiayin Chen, Michael Hush, Leonardo A. De Castro, Li Li, Per J. Liebermann, Harry J. Slatyer, Claire Edmunds, Virginia Frey, Cornelius Hempel, Alistair Milne*. Software tools for quantum control: Improving quantum computer performance through noise and error suppression. 2020.

[17] *Yuval Baum, Mirko Amico, Sean Howell, Michael Hush, Maggie Liuzzi, Pranav Mundada, Thomas Merkh, Andre R. R. Carvalho, Michael J. Biercuk*. Experimental Deep Reinforcement Learning for Error-Robust Gateset Design on a Superconducting Quantum Computer. 2021.

[18] *Pranav S. Mundada, Aaron Barbosa, Smarak Maity, Yulun Wang, T. M. Stace, Thomas Merkh, Felicity Nielson, Andre R. R. Carvalho, Michael Hush, Michael J. Biercuk, Yuval Baum*. Experimental benchmarking of an automated deterministic error suppression workflow for quantum algorithms. 2022

[19] *Andre R. R. Carvalho, Harrison Ball, Michael J. Biercuk, Michael R. Hush, Felix Thomsen*. Error-robust quantum logic optimization using a cloud quantum computer interface. 2020

[20] *Xiaofeng Gao, Zhijin Guan, Shiguang Feng, Yibo Jiang*. Quantum Circuit Template Matching Optimization Method for Constrained Connectivity. 2023

[21] *Alexander Zlokapa, Alexandru Gheorghiu*. A deep learning model for noise prediction on near-term quantum devices. 2020

[22] Pulse. Retrieved from https://qiskit.org/documentation/apidoc/pulse.html

[23] Extending the Q# Compiler. Retrieved from https://devblogs.microsoft.com/qsharp/extending-the-q-compiler/

[24] *J. M. Gambetta, A. D. Córcoles, S. T. Merkel, B. R. Johnson, J. A. Smolin, J. M. Chow, C. A. Ryan, C. Rigetti, S. Poletto, T. A. Ohki, Mark B. Ketchen, M. Steffen*. Characterization of addressability by simultaneous randomized benchmarking. Physical Review Letters 109, 24. 2012

[25] *L. Viola, E. Knill, S. Lloyd*. Dynamical decoupling of open quantum systems. Physical Review Letters 82, 12. 1999

[26] OpenQASM Live Specification. Retrieved from https://openqasm.com/intro.html#scope

[27] *A. Paetznick, K. M. Svore*. Repeat-Until-Success: Non-deterministic decomposition of single-qubit unitaries. Quantum Information & Computation 14, 15–16. 2014

[28] *S. Bravyi, A. Kitaev*. Universal quantum computation with ideal Clifford gates and noisy ancillas. Physical Review A 71, 2. 2005

[29] *Nils Herrmann, Daanish Arya, Florian Preis, Stefan Prestel.* Quantum utility -- definition and assessment of a practical quantum advantage. 2023

[30] *Nils Quetschlich, Lukas Burgholzer, Robert Wille*. Predicting Good Quantum Circuit Compilation Options. 2023

[31] MQT Predictor: Automatic Prediction of Good Compilation Paths. Retrieved from https://github.com/cda-tum/mqt-predictor

[32] Arline Benchmarks. Retrieved from https://github.com/ArlineQ/arline_benchmarks

[33] Haiqu. Retrieved from https://www.haiqu.ai/

[34] Mykola Maksymenko LinkedIn profile. Retrieved from https://www.linkedin.com/in/mykola-maksymenko-4448a839/

ВІДКРИТІ НАПРЯМИ В СТЕКУ КВАНТОВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ: ВІД NISQ ДО КВАНТОВОЇ КОРИСНОСТІ

М. Цимбаліста, М. Максименко, І. Катерняк

*Львівський національний університет імені Івана Франка,
вул. Ген. Тарнавського, 107, 79017 Львів, Україна*

ihor.katernyak@lnu.edu.ua

Покращення продуктивності квантових обчислень (QC) дозволить нам вирішувати широкий спектр складних проблем, з якими класичні комп’ютери сьогодні не можуть впоратися. Сьогодні ми відчуваємо себе ближчими до досягнення стану квантової корисності (QU), ніж будь-коли раніше. Як для академічних кіл, так і для бізнесу важливо розуміти поточний стан технологій та інструментів, точки їх розширення разом з алгоритмами оптимізації. Метою статті є надати структурований аналіз існуючого прогресу в сфері квантових обчислень. Вона слугує довідником щодо того, де очікується значний прогрес у найближчі роки, описує деталі інструментів для початку проведення експериментів і представляє еталонну архітектуру Проміжного Програмного Забезпечення Оптимізації Квантових Обчислень (QCOM) як основу, навколо якої в наступні роки очікуються нові будівельні блоки, наприклад, корпоративні з’єднувачі для певних галузей. У статті розглядається лише стек програмного забезпечення, не враховуючи можливості апаратного, оскільки вони не виглядають реалістичними на сьогодні. Все це у комбінації має допомогти вченим та інженерам визначити розумову модель того, як рухатися вперед, щоб досягти як середньострокових, так і довгострокових цілей щодо квантової корисності (QU).

Галузь квантових обчислень, яка включає апаратне забезпечення, програмне забезпечення, інструменти, алгоритми тощо, дуже складна в порівнянні з традиційними обчисленнями. База знань для галузі обмежена і в основному базується на дослідженнях і технічній документації кількох інструментів, створених лідерами галузі. Наступний прорив у квантових обчисленнях станеться на перетині алгоритмів (не алгоритмів високого рівня для вирішення конкретних проблем, скажімо, у галузі фізики, а алгоритмів на рівні стеку компіляції квантових обчислень) та існуючих інструментів (мов програмування, бібліотек програмного забезпечення або компіляторів). Ці мови програмування та компілятори мають знання про те, як маніпулювати фізичними кубітами на певному апаратному забезпеченні, переводити їх у віртуальні, виконувати виправлення помилок і надавати високорівневий інтерфейс, який приховує більшу частину складності, щоб забезпечити ефективну реалізацію квантових алгоритмів.

Існують хороші публікації про основи квантових обчислень та інструменти [1], публікації про те, що відбувається під час компіляції [2], дослідження продуктивності різних компіляторів разом із деталями інструменту, який дозволяє комбінувати кроки компіляції від різних виробників [3]. Крім того, дослідження, які показують алгоритми оптимізації розподілу кубітів [20], покращення виправлення помилок [16], тощо. Незважаючи на це, важко зробити висновок про те, як підхопити прогрес у перспективних напрямках і підходах для досягнення квантової корисності. Деталі інтерфейсів в інструментах, навколо яких можна спробувати оптимізувати продуктивність, не підсумовуються. Те саме для етапів рівня компіляції з детальною інформацією про потенційні переваги оптимізації кожної фази, разом з алгоритмами, які можуть сприяти прориву.

Дослідження змальовує картину поточного стану квантових обчислень, розглядаючи сучасні виклики та можливості, що виникають у галузі з точки зору програмного забезпечення. Концептуальна архітектура QCOM пропонується як основа реалізації програмного забезпечення для оптимізації, що охоплює різні рівні виконання квантового алгоритму від тривіальної оптимізації воріт до більш складного зіставлення кубітів, виправлення помилок і маніпуляцій з інструкціями на рівні апаратного забезпечення. Описано мотивацію оптимізації на кожному рівні разом із цікавими техніками, які в поєднанні можуть підвищити продуктивність найпопулярніших інструментів, таких як Qiskit. Аналіз допоміг визначити пріоритетні напрямки майбутніх досліджень на основі нещодавнього прогресу в галузі, а також інструменти, які могли б допомогти досягти кращого контролю за виконанням алгоритмів. Запропонована еталонна архітектура QCOM спрямована на спрощення реалізації експериментів і перевірки гіпотез у галузі.

Ключові слова: квантові обчислення, квантова корисність, продуктивність квантового алгоритму, Проміжне Програмне Забезпечення для Оптимізації Квантових Обчислень, виправлення помилок, розподіл кубітів.

Стаття надійшла до редакції 8.03.2023
Прийнята до друку 15.03.2023