Full-stack quantum computing systems in the NISQ era
Algorithm-driven and hardware-aware compilation techniques

Bandic, Medina; Feld, Sebastian; Almudever, Carmen G.

DOI
10.23919/DATE54114.2022.9774643

Publication date
2022

Document Version
Accepted author manuscript

Published in
Proceedings of the 2022 Design, Automation and Test in Europe Conference and Exhibition, DATE 2022

Full-stack quantum computing systems in the NISQ era: algorithm-driven and hardware-aware compilation techniques

Medina Bandic*, Sebastian Feld* and Carmen G. Almudever†

* Department of Quantum and Computer Engineering and QuTech, Delft University of Technology
† Technical University of Valencia

***Abstract*—The progress in developing quantum hardware with functional quantum processors integrating tens of noisy qubits, together with the availability of near-term quantum algorithms has led to the release of the first quantum computers. These quantum computing systems already integrate different software and hardware components of the so-called "full-stack", bridging quantum applications to quantum devices. In this paper, we will provide an overview on current full-stack quantum computing systems. We will emphasize the need for tight co-design among adjacent layers as well as vertical cross-layer design to extract the most from noisy intermediate-scale quantum (NISQ) processors which are both error-prone and severely constrained in resources. As an example of co-design, we will focus on the development of hardware-aware and algorithm-driven compilation techniques.**

I. Introduction

The general field of quantum computing has experienced remarkable progress in the last years becoming a tangible reality. Prototypes of quantum computers, also known as noisy intermediate-scale quantum (NISQ) computers [1], already exist and have been made available to users through the cloud [2], [3]. We will still have to wait for having large-scale and fault-tolerant quantum computers that provide the expected computational power, but the potential of this new technology is undeniable [4]–[6]. A quantum computer will not only be capable of solving relevant problems unsolvable by current classical computers, it also represents a paradigm shift in the way of how computing is performed.

Although there is still a long way to go and the challenges are diverse, huge advances have recently been made. Several experimental demonstrations of quantum computational advantage have been performed since 2019, when Google Research claimed to have achieved it on a 53-qubit programmable superconducting processor (Sycamore) [7]–[9]. Furthermore, in terms of processors' scalability, qubit counts (number of qubits on a chip) are rapidly increasing, especially in quantum technologies based on superconductors. IBM just released a 127-qubit quantum processor named Eagle [10], and expects to present a 1000-qubit chip by 2023 [11]. Note that adding more qubits exponentially increases the number of states the quantum computer can calculate with and thus its computational power.

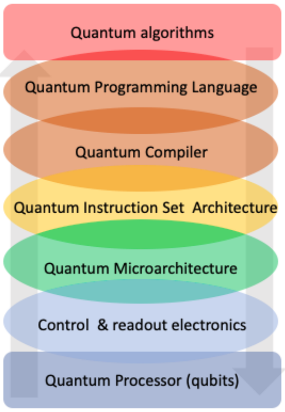

Quantum algorithms
Quantum Programming Language
Quantum Compiler
Quantum Instruction Set Architecture
Quantum Microarchitecture
Control & readout electronics
Quantum Processor (qubits)

Figure 1: Software and hardware functional elements of the quantum computing full-stack. Grey arrows represent the flow of information between hardware and software layers needed for co-design.

The progress in quantum hardware has been accompanied by advances not only on the algorithm side in the form of hybrid quantum-classical algorithms [12] for NISQ devices, but also on other required intermediate functionalities such as quantum software (i.e. programming languages and compilers) [13], instruction set architecture and microarchitecture [14]–[17] and control electronics [18]. This has lead to the development of quantum computers, as we know them today, that integrate different software and hardware components of the so-called "full-stack" [19], [20] (see also Fig 1). More precisely, full-stack quantum computing systems consist of a series of functional elements (precursors of full-fledged layers) that bridge quantum algorithms with quantum devices. Following a layered-oriented approach, which resembles classical computer architectures with some fundamental differences, quantum algorithms can be expressed using high-level programming languages and compiled to low-level instructions (e.g., quantum assembly language-instructions, QASM) that are further translated into specific signals for controlling and operating the physical qubits.

In classical computing stacks, abstractions have been introduced in the form of well-defined and self-contained layers with clear functionality that encapsulate specific information, which is only shared between adjacent layers. The level of abstraction (i.e., storing/hiding information that renders more independent layers) between different layers of the stack has been increased by virtue of the abundance of resources. This ever increasing trend resulted in software being fully independent of the underlying hardware, a desired attribute of a computer. Note though, that with the recent advent of low-power and AI co-processors, the strictly layered approach has been revisited and vertical cross-layer design and co-design are indispensable due to the scarcity of resources.

Quantum computing systems are not ready yet for such complete abstractions due to the immaturity of its hardware. To extract the most from of NISQ processors, which are severely constrained in resources (e.g. number of qubits or qubit count) and highly prone to errors, their low-level physical details need to be exposed to higher layers of the stack. This results, for instance, in compilation techniques that consider qubits' connectivity, gate error rates, error variability across the quantum device, primitive quantum gates, and crosstalk, amongst others, to efficiently execute a quantum algorithm and increase its success rate. Thus, there is a flow of information that includes relevant hardware parameters piercing bottom-up through the stack. In addition, the compiler can leverage its knowledge about the application to perform some general (e.g. gate cancellation) or even application-specific optimization on the quantum circuit. Note that the compiler's effectiveness/efficiency is key in successfully executing a quantum algorithm and, in the long term, in the adoption of this emerging technology.

Therefore, this early stage of quantum computing, in which resources are scarce, constrained and noisy, calls more than ever for a tight co-design among adjacent layers as well as vertical cross-layer design and related optimization of the full-stack [21]–[24]. Note that this co-design should occur upfront, at the conception phase of the full system. The two-fold benefit of this approach would be: i) to extract the maximum computational capability out of the constrained quantum system, but also to ii) set the precursor basis of future front-ends that will pave the way towards more layer-oriented encapsulation and abstraction.

This paper will provide an overview on the full-stack quantum computing systems in the NISQ era. To this end, we will first present the current state of quantum computers showing what functional hardware and software layers they consist of and how they bridge the gap between quantum algorithms and quantum chips. Then, we will discuss about the organization or architecture of quantum computing systems, which is still far from the layered approach used in classical computer architectures. We will emphasize that, in this early-days of quantum computing, allowing information flowing across the stack is not only a must, but also brings significant gains. Hence, a discussion on the need of co-design for optimally constructing full-stack quantum computing systems will follow. We will finally provide an example on such co-design by showing how quantum circuit mapping strategies, an integral part of the overall compilation process, can be improved and potentially increase the success rate of a quantum algorithm by being not only hardware-aware, but also application-driven.

II. Current full-stack quantum computing systems

Although quantum computers were recently developed, the idea of building one is quite old, as it was proposed in the early 80's by physicist Richard Feynman [25] for simulating quantum systems. Ten years later the first quantum algorithms appeared [12]. It was not till late 90's, when experimental demonstrations of a quantum algorithm running on a two-quit quantum processor were shown [26]. Since then, quantum hardware has substantially progressed, especially in the last years; a variety of technologies for qubit implementation, such as quantum dots, solid-state spins, trapped-ions or superconductors, are being explored [5]. In addition, its main (still) limiting characteristics are constantly improved with increased qubit counts, larger coherence times and higher gate fidelities. Note that, in spite of this progress, quantum processors are still in their infancy, being extremely resource-constrained and error-prone.

At the top of the stack, quantum algorithms have also evolved following the capabilities of quantum hardware. Most of the algorithms initially proposed (e.g., Shor, HHL) cannot be executed on current nor near-future quantum machines, as they require the incorporation of quantum error correction and fault-tolerant techniques, and therefore an immense number of qubits (in the order of millions). That is why the quantum algorithm community has made an effort to develop algorithms suitable for NISQ processors [12].

To close the gap between quantum algorithms and devices, and to be able to run quantum algorithms on larger and larger quantum processors, further components needed to be added and integrated to complete the system such as the software part, the interface between software and hardware, and the control stack. This led to the development of modern full-stack quantum computing systems, which were inspired by classical computer architectures, and include the following elements (see Fig. 1): quantum applications, high-level quantum programming languages and compilers, quantum instruction set architecture and related microarchitecture, control electronics and the quantum device. Although essential functional elements of the quantum computing full-stack have been identified and integrated, the architecture and organization of such stacks may differ between systems and is in constant evolution as the field progresses [20].

For current quantum computers, algorithms can be programmed using programming frameworks [13], [27]. They not only feature high-level programming languages such as Python to describe quantum circuits, but also compilers that translate those high-level instructions into low-level ones, usually expressed in a quantum assembly-like language understandable and therefore executable by the underlying processor. As further explained in Sec. III, quantum compilers are also responsible for making transformations to the quantum circuit to fulfill the quantum hardware's constraints, along with optimizations to reduce the circuit depth, for instance. Note that quantum compilers are located in the middle of the stack, as they are key to bridge quantum applications to quantum devices. The output of the compiler, low-level instructions, are then further translated into specific pulses to operate and control the chip's qubits. For this purpose, quantum instruction set architectures [14], [15] and microarchitectures [16], [17] as well as control electronics (at even cryogenic temperatures) [18] have been developed.

Towards increased abstraction in quantum computing systems: the need for co-design to architect

Classical computers consist, from an architectural point of view, of a series of layers with well-defined functionality in which relevant information is encapsulated and only exposed to adjacent levels. This layered architecture has evolved with a growing level of abstraction between layers as the amount of resources has been substantially increased.

Most of the functional elements present in classical systems, already in the form of layers, can also be identified in current quantum computers, but not as mature though. In full-stack quantum computing systems, there is still a tight interplay among and across functionalities and therefore no strict layered abstracted architecture yet with clear allocation of functionality per layer exists. This is mainly due to severely limited and impaired resources in quantum devices, which encompass: limited qubit count and connectivity among them, short coherence time, high operation error rates, and crosstalk. Actually, allowing information from the physical lowest-level (i.e. quantum processor) to flow up to high-level functions is not only a must for being able to (successfully) execute an algorithm, but the best way to maximize performance [23]. A notable example of such information flow is within the compiler, which uses quantum processor characteristics to modify the circuit for meeting the hardware's constraints but also to maximize the corresponding success rate. Instances of that include the introduction of software techniques to deal with or alleviate crosstalk [28], [29] and noise-aware compilation methods [30]. Note that quantum hardware information can be combined with algorithm parameters to further optimize compilation techniques given a specific application (application-specific compilers) [24], [31].

These examples, in which information exchange is bounded, punctual and limited, constitute ad-hoc predecessors of full co-design techniques. As defined in [22], co-design refers to "the flow of information between different hardware and software stack layers, in order to improve the overall application execution and hardware design. The information flow might include: key hardware parameters, design specifications, and resource requirements up and down the stack. Co-design for quantum computing is about incorporating this information into the techniques and system designs at every layer of the stack to make optimal use of limited resources". Culminating co-design, both across adjacent layers as well as cross-layer vertical design, will require these techniques to be all-pervasive in coverage, information-rich in exchange, and structured in their application [21]. Finally, we postulate that co-design is not an aim per se, but a means to eventually achieve full abstraction, since this information exchange across layers will serve as a basis to implement front-ends that allow self-contained encapsulated stack layers.

The next section will provide a specific example of quantum co-design. More precisely, it will show how the mapping process, which is an essential part of the compiler, can be further optimised by considering not only hardware characteristics, but also relevant algorithm properties.

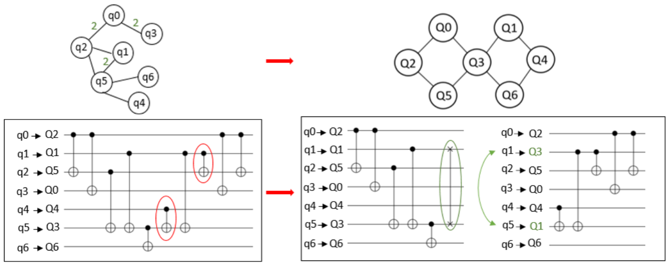


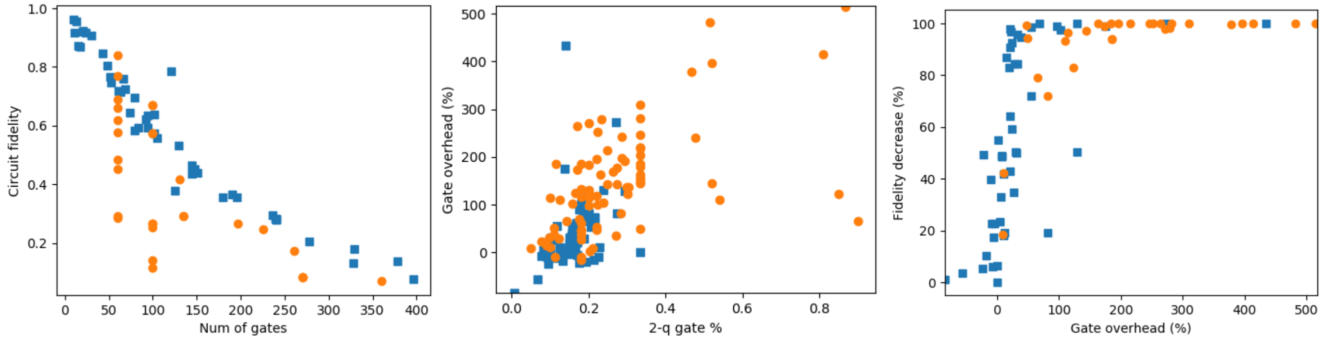
Figure 2: Running a quantum circuit on a Surface-7 quantum processor [32]. Top-left: Interaction graph of the circuit shown below. Top-right: The chip coupling graph; nodes represent physical qubits and edges show connections on the chip (possible interactions). Bottom: Qubits in the circuit (qi) are mapped onto physical qubits labeled with Qi. An extra SWAP gate is required for being able to perform all CNOT gates. [33]

III. Quantum circuit mapping: Accommodating quantum algorithms to resource-constrained quantum devices

As discussed above, quantum computing full-stacks have been developed to enable the execution of quantum algorithms on quantum processors. Current NISQ devices can only handle simpler algorithms, in terms of number of gates and circuit depth, as they are still constrained by the presence of noise: quantum gates have high error rates and qubits are fragile and decohere over time resulting in information loss. On top of that, running an algorithm on a NISQ device is not a straightforward process, as there are other hardware limitations that must be considered.

One of the most relevant quantum hardware constraint is *limited qubit connectivity*. For most technologies, including superconducting qubits and quantum dots, qubits are arranged in a 2D grid topology (see Fig. 2, top right) allowing only nearest-neighbor interactions. This means that in order to perform a two-qubit gate, the two interacting qubits have to be placed in neighboring locations on the chip, which is not always possible (Fig. 2). Other constraints that affect algorithm execution are: i) *primitive gate set*, that is, the quantum gates supported by the device. Note that a quantum chip gate set does not necessarily have to match the one used in the circuit to be run; and ii) *classical control constraints* that come from the use of shared control electronics required to scale up quantum computing systems. This limits the operations' parallelization. It is the responsibility of the compiler, and more precisely of *the mapping process*, to satisfy all device constraints while accommodating the algorithm's needs.

The quantum mapping process consists of the following steps (not necessarily in this particular order), see also Fig. 2: 1) *Decomposition of the gates* of the circuit to the primitive gate set; 2) *Scheduling* quantum operations to leverage parallelism and therefore shorten execution time; 3) Smartly *placing virtual qubits* (from the circuit) *onto physical qubits* (placements on actual chip) such that the previously mentioned nearest-neighbor two-qubit gate constraint is satisfied as much as possible during circuit execution; and 4) *Routing* or exchanging positions of virtual qubits on the chip such that all qubits that need to interact during circuit execution



(a) **(b)** **(c)**

Figure 3: Impact of the circuit mapping process. (a) Gate number vs. circuit fidelity, (b) 2-qubit gate % influence on gate overhead, and (%) (c) Gate overhead and decrease in fidelity relation, when randomly generated circuits (orange circles) and real algorithms (blue squares) taken from [34] are mapped into an extended 100-qubit version of the Surface-17 hardware configuration. To this purpose, the trivial mapper integrated in the OpenQL compiler [27] has been used. For a) and c) only circuits with less than 400 gates were used. Circuit fidelity is calculated as product of fidelities for all one- and two- qubit gates in the circuit, based on the error-rate values taken from [32].

are adjacent. This is done by inserting additional quantum gates called SWAPs, whose purpose is to exchange virtual qubit values (i.e., quantum states) between physical qubits on the chip. Therefore, the resulting circuit after-mapping might have more gates and higher depth, and hence a longer execution time than the original one. Due to the previously mentioned highly-erroneous quantum operations and qubit decoherence, it is crucial to create only minimal overhead as it affects the algorithm's fidelity: as the number of gates in the circuit increases, the fidelity decreases (see Fig. 3(a)).

There exist various approaches to solve the mapping problem, each using different methods and strategies [35]–[42]. They also differ in the cost function optimized in the mapping process, which is also used as the performance metric to assess the mapper. Usual metrics are gate overhead (number of SWAPs), circuit depth and latency overhead (number of time-stamps) and reliability/fidelity or success rate probability. Fig. 3(b) shows how the percentage of two-qubit gates in the circuit affects the gate overhead: the higher this percentage, the more qubits interact and therefore the higher the gate overhead caused by routing. Additional (SWAP) gates then impact the circuit's fidelity as shown in Fig. 3(c).

The majority of circuit mapping approaches mostly focus on device characteristics without considering structural properties of the quantum circuit itself. When characterizing benchmark circuits, the only parameters taken into account in literature are gate and qubit count, two-qubit gate percentage (Fig. 3) and at times circuit depth. More in-depth algorithm characterization is of crucial importance because it enable enable us to: i) perform an exhaustive comparison and taxonomy of algorithms; ii) perform an in-depth analysis on the performance of compilation (mapping) techniques, and iii) further improve the compilation process creating not only hardware-aware but also algorithm-driven solutions.

Some works have already pointed out the importance of considering algorithm properties [43]–[45] for further improving the compilation techniques and, in particular, interaction graphs for the mapping procedure [33], [46]. *Interaction graphs* are graphical representations of the two-qubit gates of a given quantum circuit. Fig. 2 shows an example of a quantum circuit along with its interaction graph representation. Edges represent two-qubit gates and nodes are the qubits that participate in those. If a circuit comprises multiple two-qubit gates between pairs of qubits, it results in a weighted graph (like in Fig. 2) which shows how often each pair of qubits interacts and how those interactions are distributed. This additional information can be leveraged to provide more insight into the structure of a circuit that is otherwise hidden when only considering common algorithm parameters such as number of qubits and gates and two-qubit gate percentage. To illustrate this, Fig. 4 shows the interaction graphs of two quantum algorithms, a real one (QAOA, on the left) and a randomly generated circuit (on the right), with the same properties when only characterized in terms of the three common algorithm parameters. What can be noticed is that their interaction graph structure is quite different: the graph of the random circuit is more complex with full-connectivity and present a different distribution of the interactions between qubits, that is, of the weights. This will result in more routing and therefore higher overhead. As shown in Fig. 3, the gate overhead and fidelity decrease is, on average, higher for synthetic (random) algorithms than for the real ones.

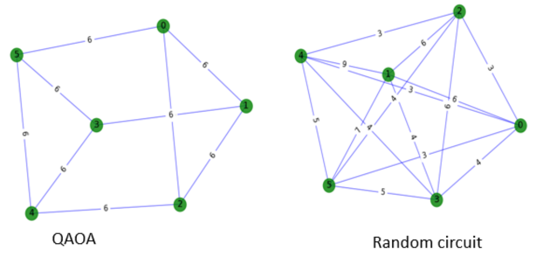


Figure 4: Interaction graphs of circuits with the same size parameters: num. of qubits = 6, num. of gates = 456, 2-qubit gate % = 0.135.

Therefore, analysing interaction graphs might help us understand why a mapping solution works better for specific (groups) of algorithms first, and then come up with optimised mapping techniques that are both, algorithm-driven and hardware-aware. Algorithm-driven devices could be an effective solution in dealing with limited NISQ computing resources [22], [45], as they can precisely be designed for some dedicated purpose. Even in classical computing used in daily lives, different computing assets are required for different specific purposes (applications).

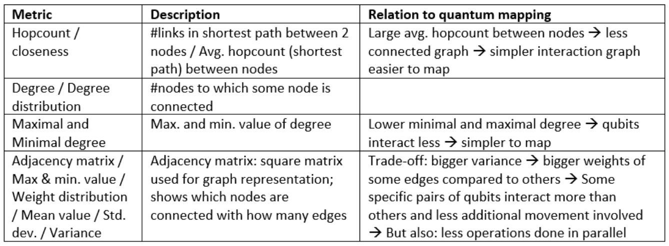
IV. Algorithm-driven mapping solutions: characterising qubit interaction graphs

As explained in the previous section, we will broaden the scope of algorithm characterization by introducing interaction-graph-based profiling. We will discuss how graph parameters might be meaningful for improving quantum circuit mapping techniques. Some preliminary results will be presented, showing the relation of those graph parameters to the performance of circuit mapping.

The main reason and importance of exploring the properties of interaction graphs is the fact that they represent the core constraint that needs to be dealt with during the mapping process: two-qubit gates and their dispersion among pairs of qubits. For that purpose we took input from graph theory and characterized quantum algorithms based on their interaction graph metrics [47] such as average shortest path, connectivity, clustering coefficient and similar ones, with a focus on metrics that are of interest for the mapping problem.

What can be noticed is that large number of handpicked, mapping-related metrics is codependent, i.e. they scale in the same manner. In order to reduce the parameter space and select only features that are necessary, a Pearson correlation matrix was created. Applying this method reduced our previous metric set to: average shortest path (hopcount/clossenes), maximal and minimal degree and adjacency matrix standard deviation, as shown in Tab. I. Using this new metrics and the common circuit parameters, algorithms can be clustered based on their similarities. Ideally, quantum algorithms with similar properties are ought to show similar performance when run on specific chips using a given mapping strategy.

Some preliminary results on the use of graph-based metrics, with the final aim to further optimise the quantum circuit mapping process, are shown in Fig. 5. We have compiled 200 quantum circuits by using the same hardware and mapping configuration as described in caption of Fig. 3. The benchmark set used [34] contains circuits of a large variety in size (1-54 qubits, 5-100000 gates, 10-90% two-qubit gate percentage) and type (random, reversible ones [48] and those corresponding to real algorithms). Our first goal was to analyse how the graph-based metrics relate to gate overhead and consequently algorithm fidelity decrease.



Metric	Description	Relation to quantum mapping
Hopcount / closeness	#links in shortest path between 2 nodes / Avg. hopcount (shortest path) between nodes	Large avg. hopcount between nodes → less connected graph → simpler interaction graph easier to map
Degree / Degree distribution	#nodes to which some node is connected	
Maximal and Minimal degree	Max. and min. value of degree	Lower minimal and maximal degree → qubits interact less → simpler to map
Adjacency matrix / Max & min. value / Weight distribution / Mean value / Std. dev. / Variance	Adjacency matrix: square matrix used for graph representation; shows which nodes are connected with how many edges	Trade-off: bigger variance → bigger weights of some edges compared to others → Some specific pairs of qubits interact more than others and less additional movement involved → But also: less operations done in parallel

Table I Metrics for characterizing interaction graphs [47] and their relation to mapping.

Fig. 5 shows that all circuits with high gate overhead had on average low variation in edge weight distribution, low average shortest path between qubits and higher max. degree, which are expected values from Tab. I.

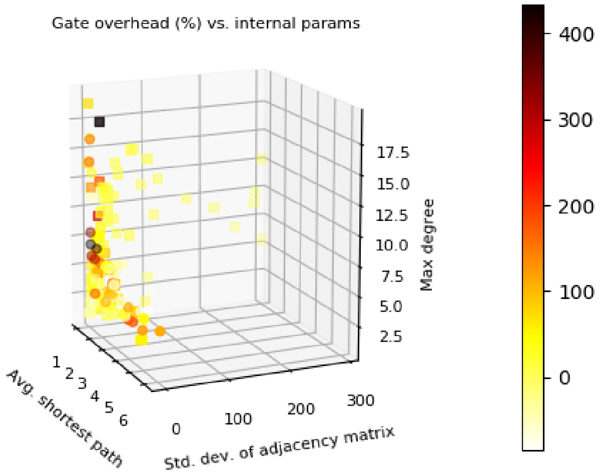


Figure 5: Gate overhead (%) vs. inter. graph parameters. Each point represents a benchmark mapped on the chip. Synthetically generated circuits are marked with a circle, real ones with a square.

V. Conclusions

Current intermediate-scale quantum computers already integrate the different hardware and software elements of the so-called full-stack, allowing the execution of quantum algorithms on NISQ devices. In this early-stage of quantum computing, quantum processors are resource constrained and highly error-prone, which is preventing these stacks from following an abstracted layered approach as used in classical computers. In terms of computer architecture, there is still a tight interplay among and across layers, which shows a progressive and continuously changing allocation of functionalities and requires the addition of new ones as the field advances. In other words, the organization of the stack is in constant evolution and it is being shaped based on the quantum hardware capabilities as well as on the quantum algorithms requirements. A higher level of abstraction between layers is expected as the technology matures.

A crucial aspect for architecting not only nowadays but also near-term quantum computing full-stacks, which are expected to be in the form of application-specific quantum accelerators, is co-design. Allowing information flowing up and down across the stack is key to optimise the stack and extract the maximum computational power out of the constrained quantum system. Although co-design techniques are already being used in the development of, for instance, quantum compilers or even quantum algorithms, applying co-design in its broadest sense, will require these techniques to be all-pervasive in coverage, information-rich in exchange, and structured in their application. We postulate that co-design will be also a mean to eventually achieve full abstraction, since this vertical information exchange across the stack will help to implement front-ends allowing the development of self-contained encapsulated layers.

Finally, quantum compilers, and more precisely the mapping of quantum circuits, are the most sensitive parts of the stack for co-design as their main functionality is to make some transformations to the quantum circuits that they can be efficiently run on a given quantum device. To further improve the compilation techniques is key for achieving higher algorithm success rates in the NISQ era and therefore, towards showing "quantum practicality or usefullness”. The optimization of the mapping methodologies, requires them to be not only hardware-aware but also algorithm-driven. This calls for a more in-depth profiling of quantum applications. We show instances that looking into the qubit interaction graph and considering graph-based metrics might assist, guide, dimension and optimize such mapping techniques.

Acknowledgment

The authors sincerely appreciate scientific discussions with Prof. Eduard Alarcon (UPC).

References

[1] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[2] IBM. Quantum experience, 2017.

[3] QuTech. Quantum inspire, 2020.

[4] Matthias Möller and Cornelis Vuik. On the impact of quantum computing technology on future developments in high-performance scientific computing. *Ethics and Information Technology*, 19(4):253–269, 2017.

[5] Salonik Resch and Ulya R Karpuzcu. Quantum computing: an overview across the system stack. *arXiv preprint arXiv:1905.07240*, 2019.

[6] Jean-François Bobier et al. What happens when ‘if’ turns to ‘when’ in quantum computing? *Boston Consulting Group, July*, 2021.

[7] Frank Arute et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[8] Han-Sen Zhong et al. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020.

[9] Yulin Wu et al. Strong quantum computational advantage using a superconducting quantum processor. *Physical review letters*, 127(18):180501, 2021.

[10] IBM. Ibm quantum processor eagle, 2021.

[11] IBM. Ibm’s roadmap for scaling quantum technology, 2020.

[12] Kishor Bharti et al. Noisy intermediate-scale quantum (nisq) algorithms. *arXiv preprint arXiv:2101.08448*, 2021.

[13] Frederic T Chong et al. Programming languages and compiler design for realistic quantum hardware. *Nature*, 549(7671):180–187, 2017.

[14] Xiang Fu et al. eqasm: An executable quantum instruction set architecture. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 224–237. IEEE, 2019.

[15] Xiang Zou et al. Enhancing a near-term quantum accelerator’s instruction set architecture for materials science applications. *IEEE Transactions on Quantum Engineering*, 1.

[16] Xiang Fu et al. An experimental microarchitecture for a superconducting quantum processor. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 813–825, 2017.

[17] Mengyu Zhang et al. Exploiting different levels of parallelism in the quantum control microarchitecture for superconducting qubits. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 898–911, 2021.

[18] Xiao Xue et al. Cmos-based cryogenic control of silicon quantum circuits. *Nature*, 593(7858):205–210, 2021.

[19] Carmen G. Almudever et al. The engineering challenges in quantum computing. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 836–845. IEEE, 2017.

[20] Margaret Martonosi and Martin Roetteler. Next steps in quantum computing: Computer science’s role. *arXiv preprint arXiv:1903.10541*, 2019.

[21] Carmen G Almudever and Eduard Alarcon. Structured optimized architecting of full-stack quantum systems in the nisq era. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 762–767. IEEE, 2021.

[22] Teague Tomesh and Margaret Martonosi. Quantum codesign. *IEEE Micro*, 41(5):33–40, 2021.

[23] Yunong Shi et al. Resource-efficient quantum computing by breaking abstractions. *Proceedings of the IEEE*, 108(8):1353–1370, 2020.

[24] Gushu Li et al. Software-hardware co-optimization for computational chemistry on superconducting quantum processors. *arXiv preprint arXiv:2105.07127*, 2021.

[25] Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6):467–488, 1982.

[26] Isaac L Chuang et al. Experimental implementation of fast quantum searching. *Physical review letters*, 80(15):3408, 1998.

[27] Nader Khammassi et al. Openql: A portable quantum programming framework for quantum accelerators. *arXiv preprint arXiv:2005.13283*, 2020.

[28] Prakash Murali et al. Software mitigation of crosstalk on noisy intermediate-scale quantum computers. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1001–1016, 2020.

[29] Yongshan Ding et al. Systematic crosstalk mitigation for superconducting qubits via frequency-aware compilation. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 201–214. IEEE, 2020.

[30] Prakash Murali et al. Full-stack, real-system quantum computer studies: Architectural comparisons and design insights. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 527–540. IEEE, 2019.

[31] Lingling Lao and Dan Browne. 2qan: A quantum compiler for 2-local qubit hamiltonian simulation algorithms. *arXiv preprint arXiv:2108.02099*, 2021.

[32] R Versluis et al. Scalable quantum circuit and control for a superconducting surface code. *Phys. Rev. Applied*, 8(3):034021, 2017.

[33] Medina Bandic et al. On structured design space exploration for mapping of quantum algorithms. In *2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–6. IEEE, 2020.

[34] qbench benchmark suite. https://github.com/QE-Lab/qbench, 2021.

[35] Carmen G Almudever et al. Realizing quantum algorithms on real quantum computing devices. *arXiv preprint arXiv:2007.01000v1*, 2020.

[36] Lingling Lao et al. Timing and resource-aware mapping of quantum circuits to superconducting processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[37] Matteo G Pozzi et al. Using reinforcement learning to perform qubit routing in quantum compilers. *arXiv preprint arXiv:2007.15957*, 2020.

[38] Stefan Hillmich et al. Exploiting quantum teleportation in quantum circuit mapping. In *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 792–797. IEEE, 2021.

[39] Toshinari Itoko et al. Optimization of quantum circuit mapping using gate transformation and commutation. *Integration*, 70:43–50, 2020.

[40] Bochen Tan and Jason Cong. Optimal qubit mapping with simultaneous gate absorption. *arXiv preprint arXiv:2109.06445*, 2021.

[41] Hui Jiang et al. Quantum circuit transformation based on subgraph isomorphism and tabu search. *arXiv preprint arXiv:2104.05214*, 2021.

[42] Sanjiang Li et al. Qubit mapping based on subgraph isomorphism and filtered depth-limited search. *IEEE Transactions on Computers*, 2020.

[43] Thomas Lubinski et al. Application-oriented performance benchmarks for quantum computing. *arXiv preprint arXiv:2110.03137*, 2021.

[44] Daniel Mills et al. Application-motivated, holistic benchmarking of a full quantum computing stack. *arXiv preprint arXiv:2006.01273*, 2020.

[45] Gushu Li et al. Towards efficient superconducting quantum processor architecture design. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1031–1045, 2020.

[46] Matthew Steinberg et al. A noise-aware qubit mapping algorithm evaluated via qubit interaction-graph criteria. *arXiv preprint arXiv:2103.15695*, 2021.

[47] Javier Martın Hernández and Piet Van Mieghem. Classification of graph metrics. *Delft University of Technology: Mekelweg, The Netherlands*, pages 1–20, 2011.

[48] Robert Wille et al. Revlib: An online resource for reversible functions and reversible circuits. In *38th International Symposium on Multiple Valued Logic (ismvl 2008)*, pages 220–225. IEEE, 2008.