Alwin Zulehner Follow
Sep 12 · 5 min read

How Computer-Aided Design helped me winning the QISKit Developer Challenge

Quite often, people ask me what the actual topic of my PhD studies is. Then, I start to talk about quantum computers and how they can change the world in the not so far future, about qubits and quantum-physical phenomena like superposition and entanglement, etc. However, this is only half the truth. While indeed *quantum computing* is my main focus, I am approaching it from a different direction than many of the peers in the field: I (together with my supervisor Robert Wille) am trying to bring Computer-Aided Design (CAD) into this domain. And, indeed, although quantum computing poses some severe challenges to be solved, we strongly believe that at least some of them can nicely be solved using CAD methods.

In this blog post, I want to emphasize the need for CAD methods in the field of quantum computing and show how my background in CAD helped to win the QISKit Developer Challenge. By this, I want to motivate people to join this interesting niche since CAD methods are required more and more. This is especially caused by the fact that quantum computers are getting real and their fidelity, coherence time, and the number of available qubits steadily increase. Solving certain design problems by hand becomes infeasible and, thus, requires automatic methods.

But one by one: At the beginning of 2018, IBM started several challenges to encourage people to take advantage of the IBM Q Experience and the IBM QISKit development platform. One of them was the IBM QISKit Developer Challenge, which asked for a compiler that maps circuits composed of random operations from SU(4) to a certain quantum hardware topology. The major parts of such a compiler are (1) to break down the functionality to elementary operations and (2) to find a mapping of the logical qubits (from the circuit) to the physical qubits available in the actual quantum computer.

Especially the second part is crucial, since certain operations can only be conducted on certain pairs of physical qubits (e.g. Figure 1 shows a so-called coupling map, which defines what pairs of physical qubits can actually interact in operations realized on the IBM QX5 architecture).

In order to respect these constraints, so-called SWAP operations are frequently applied that exchange the state of two physical qubits and, by this, move logical qubits so that the desired operations can be realized. Figure 2 illustrates the process: First (left-hand side of Figure 2), the desired quantum circuit is shown which includes several operations over qubits which are not allowed according to the coupling map in Figure 1. But adding SWAP operations as shown in the right hand side of Figure 2 moves all the qubits to positions so that they eventually are in line with the restrictions imposed by the coupling map. However, in order to keep the costs low and to increase the fidelity of the system, the number of these SWAP operations should be kept as small as possible—a typical CAD problem.

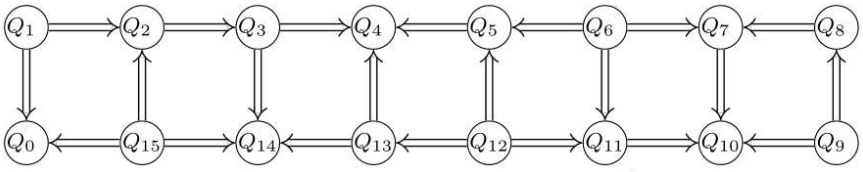

Figure 1: Coupling map of IBM QX5 — Only operations which work on connected qubits can be realized

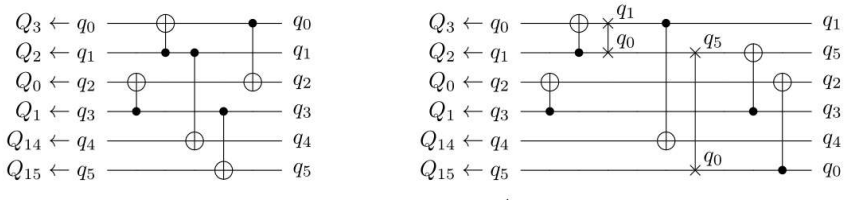

Figure 2: The original circuit (left) and how to satisfy the restrictions from the coupling map (right)

Actually, the problem was not completely new to me, since in my PhD studies I had already developed such a mapping algorithm for IBM's QX architectures (which I also incorporated into QISKit). However, it turned out that the kind of random circuits considered in the challenge represent a worst case scenario for this mapping algorithm, since the utilized search algorithm explores a too large part of the exponential search space. Thus, I started to analyze the circuits considered in the developer challenge and why they cause problems on my previous algorithm. By this, I came up with a new idea that circumvents the bottlenecks of my previous mapping approach and exploits the characteristics of the considered circuits to reduce the caused overhead. Moreover, I did not only focus on the mapping itself (as I did before), but also on a dedicated pre-processing and post-mapping optimization to provide a fully-fletched compiler.

The three stages work as follows:

First, a pre-processing stage is conducted to reduce the complexity (i.e. the number of operations) of the circuit to be mapped. This pre-processing was motivated by the structure of the circuits which allows for a dedicated grouping of operations. Adding SWAP operations for entire groups rather than several times for single operations significantly reduces complexity and costs.

Afterwards, the actual mapping problem is considered in the second stage. After determining an (arbitrary) initial mapping, the actual mapping procedure is composed of two alternating steps: First, the respective operations of all groups which already satisfy the restrictions and dependencies are added to the circuit. Then, the set of groups that could be added next (according to their precedence in the circuit) is determined. Since they all do not satisfy the restrictions yet, an A* method (an efficient solution which is common in the CAD domain) determines the best possible sequence of SWAP operations such that the constraints are satisfied for at least one of these groups.

Finally, the resulting circuit now still provides significant potential for optimization due to the resulting (re-grouped) operations and SWAP operations. To exploit that, a post-mapping process is applied which basically re-compiles the groups again—yielding to further significant cost reductions. This especially works well, when applying a SWAP gate operation to two qubits, to which an operationfrom SU(4) has been applied right before. In this case, the SWAP operation is free as shown in Figure 5.

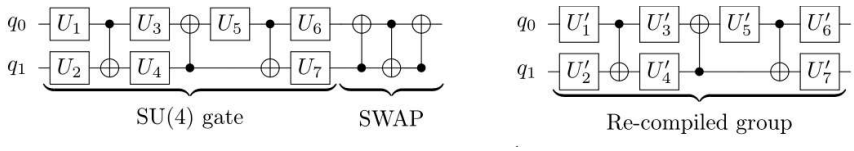

Figure 3: Reducing the costs by a post-mapping process

Overall, all three steps basically incorporated basic CAD methods which just have nicely been employed for this particular problem. An open-source implementation of the resulting method is available at http://iic.jku.at/eda/research/ibm_qx_mapping. There, you will also find papers providing a much more in-depth treatment of the methods. Evaluations conducted during the challenge showed that the resulting compiler does not only consistently produce circuits with at least 10 percent better cost than the competition, but also was more than 6 times faster than the others (according to IBM).

I hope that this blog post helps to get a an idea of the main ideas of the proposed method (as well as that CAD indeed is beneficial for quantum computing). I am honored that the jury awarded this submission with the first prize of the QISKit Developer Challenge and would like to sincerely thank everybody involved.