Breaking Down Quantum Compilation: Profiling and Identifying Costly Passes

Felix Zilk, Alessandro Tundo, Vincenzo De Maio, Ivona Brandic
HPC Research Group, Faculty of Informatics
TU Wien, Vienna, Austria
{felix.zilk,alessandro.tundo,vincenzo.maio,ivona.brandic}@tuwien.ac.at

***Abstract*—With the increasing capabilities of quantum systems, the efficient, practical execution of quantum programs is becoming more critical. Each execution includes compilation time, which accounts for substantial overhead of the overall program runtime. To address this challenge, proposals that leverage precompilation techniques have emerged, whereby entire circuits or select components are precompiled to mitigate the compilation time spent during execution. Considering the impact of compilation time on quantum program execution, identifying the contribution of each individual compilation task to the execution time is necessary in directing the community's research efforts towards the development of an efficient compilation and execution pipeline. In this work, we perform a preliminary analysis of the quantum circuit compilation process in Qiskit, examining the cumulative runtime of each individual compilation task and identifying the tasks that most strongly impact the overall compilation time. Our results indicate that, as the desired level of optimization increases, circuit optimization and gate synthesis passes become the dominant tasks in compiling a Quantum Fourier Transform, with individual passes consuming up to 87% of the total compilation time. Mapping passes require the most compilation time for a GHZ state preparation circuit, accounting for over 99% of total compilation time.**

***Index Terms*—quantum computing, quantum programs, quantum circuit compilation, profiling, Qiskit**

I. Introduction

Quantum computing (QC) has gained particular interest in both academia and industry due to its promise to significantly speed up certain computational tasks [1], [2]. Potential applications include the simulation of physical systems in materials science and chemistry [3], optimization problems [4], and machine learning [5]. Over the last decade, QC has become particularly popular due to significant advances that brought early prototype in-lab demonstrations [6], [7] to production-grade systems available as cloud-based services [8], [9] or integrated into high-performance computing (HPC) facilities [10], [11]. To date, researchers have used these state-of-the-art quantum systems to run computational workloads involving more than 100 qubits [12], [13] and demonstrate computational advantage for specific problems using different physical platforms [14], [15].

Dedicated software development kits (SDKs), such as Qiskit [16], are used for developing quantum algorithms and programming quantum devices. Their current workflow for the development and execution of quantum programs includes several steps, including high-level optimization, compilation, execution on hardware, and post-processing tasks [17], [18]. The compilation step, that is, the transformation of abstract quantum programs into instructions that can be executed on a quantum computer, involves numerous computationally expensive tasks, such as mapping logical qubits from an abstract circuit definition to a physical implementation on the quantum device and converting high-level circuit operations into native hardware instructions [17], [18]. By default, quantum programs are entirely compiled at each execution. Consequently, the time spent on compilation has a significant contribution to the overall runtime of the whole program [9], [19], which leads to a significant runtime overhead, especially as both the size and complexity of the original circuit scale [19].

Recently, a number of solutions have been proposed to address this challenge through the use of precompilation methods, including [9], [19]–[23], which involve the compilation of a portion of the source code to be executed prior to deployment and stored in advance, rather than being compiled at runtime. These approaches concentrate on precompiling specific gates [23], the logical circuit level [19], or the pulse level [21], [22]. However, the question of which individual tasks, so-called passes, and which parameters (e.g., circuit structure, optimization level, etc.) contribute to compilation time — and to what extent — has not been addressed.

In this work, we perform a preliminary analysis of Qiskit's built-in compiler toolchain to identify which of its passes affect the compilation time for a given circuit with varying optimization levels. Consequently, we aim to help researchers and developers identify potential bottlenecks in the compilation process and to effectively use precompilation techniques. In particular, we comprehensively profile Qiskit's preset compiler pipelines, focusing on the cumulative CPU time required for individual passes and identifying the top 10 most expensive passes for a Quantum Fourier Transform (QFT) [24] and a Greenberger-Horne-Zeilinger (GHZ) state [25] preparation circuit with 100 qubits each. Our results show that while the contributions of synthesis passes among the top 10 reveal a comparable trend for both circuits and all optimization levels, especially for higher optimization levels, the impact of qubit mapping and circuit optimization passes varies significantly between the two circuits. For instance, a single mapping pass accounts for over 99% of the total compilation time when compiling GHZ with optimization levels 2 and 3. Similarly, for optimization level 3 and QFT, a single circuit optimization pass accounts for $\approx$87% of the overall compilation time.

II. Background

A. Quantum Program Execution Model

The execution of a quantum program on a quantum computer can be modeled in a workflow consisting of several steps, as outlined in [18] and illustrated in Fig. 1. First, a suitable algorithm is selected to solve the problem at hand. Next, a hardware-independent circuit optimization step is applied, followed by a hardware selection step [17]. Once a device has been selected and the physical constraints of the targeted quantum processing unit (QPU) are known, device-specific quantum circuit compilation can proceed [17]. In this phase, the circuit is compiled and optimized for the selected hardware platform on which it is to be executed. Finally, the compiled circuit is sent to a quantum computing device for deployment and execution, which returns the result [18].

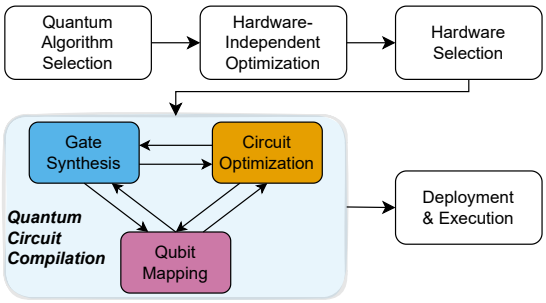


Fig. 1: The depicted workflow for quantum program execution is based on the model in [18] and adapted with a more nuanced view of quantum circuit compilation, as presented in [17].

Quantum circuit compilation (QCC) — i.e., the transformation of abstract, hardware-agnostic quantum circuits into a version optimized for the target hardware — is a non-trivial procedure involving numerous time- and resource-intensive tasks. Specifically, the abstract quantum circuit must be translated into a sequence of instructions that can be executed natively on the target QPU; a task known as gate synthesis [17]. In addition, the hardware-agnostic qubit layout of the abstract quantum circuit has to match the specific topology of the physical device; a process termed qubit mapping, a known NP-complete problem [26]. Both processes change the composition of the circuit, leading to further possibilities for circuit optimization (see Fig. 1).

B. Quantum Circuit Compilation in Qiskit

Qiskit performs QCC by executing a sequence of compilation tasks, referred to as passes. These passes are typically executed through a `PassManager`, which arranges and performs a series of circuit inspections and transformations [27]. Specifically, Qiskit offers four built-in compiler pipelines, each designed to cater to distinct levels of optimization: 0 (no optimization), 1 (light optimization), 2 (medium optimization), and 3 (high optimization). Each of these built-in compiler pipelines is organized into multiple stages: initialization, layout, routing, translation, optimization, and scheduling.

These stages correspond to the steps shown in Figure 1, in particular, hardware-independent optimization (*initialization*), qubit mapping (*layout* and *routing*), gate synthesis (*translation*), and circuit optimization (*optimization*).

The pipelines are generated by the `generate_preset_passmanager` function, which creates the respective `PassManager`. According to the Qiskit documentation, the standard usage of this method requires at least a `backend` instance and an `optimization_level` as inputs to generate the pipeline; hence, we will focus on these two variables in our preliminary analysis. The key method that executes the logic of each compiler pass is the `run(dag)` method, which is implemented by each individual pass. The execution of the entire chain of compiler passes is implemented by the pass manager's `run(circuit)` method, which applies all involved passes to the quantum circuit and returns a compiled circuit that is executable on the target hardware.

III. Related Work

Existing work on quantum compilation [9], [19], [21]–[23] concentrates primarily on the precompilation of either entire logical circuits or parts thereof with the objective of reducing compilation time during execution later on. Kudrow et al. [23] propose a method that aims to reduce the compilation time for arbitrary rotation gates by precompiling a specified set of rotations. In contrast, Gokhale et al. [22] and Cheng et al. [21] employ precompilation to mitigate compilation overhead for approaches that compile abstract circuits directly to the pulse level. Karelakas et al. [9] have addressed the runtime bottleneck that arises from QCC in quantum-classical cloud architectures, focusing on hybrid variational algorithms. Their proposal includes a compilation method that precompiles an abstract circuit with placeholders for gate parameters, bypassing the need to recompile the entire circuit at each iteration. Finally, the work of Quetschlich et al. [19] has proposed a compilation approach that precompiles a generic quantum circuit that represents an entire class of problems. Their approach adapts the compiled circuit to the specific problem at runtime by solely subtracting unnecessary gates. However, while these approaches aim to reduce the required compilation time during quantum program execution, they do not consider the contribution of individual compiler passes.

Recently, the Qrisp [28] framework proposed caching for individual functions of a quantum program. Here, the Python interpreter traces a decorated function only once, ensuring that subsequent calls to the same function execute without interpreter-induced delay. However, there is currently no guideline available to determine which functions are more or less suitable for this approach.

These contributions underscore the significance of the required compilation time for quantum programs. Nevertheless, the existing literature is missing an examination of which individual passes impact the overall compilation time of a given circuit, taking into account their dependencies on parameters such as circuit structure and desired degree of optimization.

IV. Profiling Methodology

Figure 2 illustrates our profiling methodology to monitor all compiler passes involved in a Qiskit program execution and collect their execution profile[1]. We choose Qiskit due to its wide adoption as an SDK for quantum programs [30], and both its flexibility and performance, as reported in a recent benchmark study [31]. Specifically, we conduct our analysis using Qiskit SDK v1.3.2 and its preset compilation pipelines.

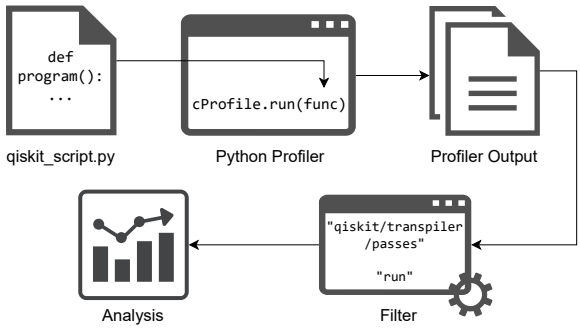


Fig. 2: The workflow of our profiling methodology.

We monitor the program execution using the Python profiler cProfile [29]. The profiler output provides a set of statistics, including the cumulative time spent in a function and calls to all its sub-functions. From this output, we filter only for those functions that contain the target location where Qiskit's built-in passes are implemented, i.e., "`qiskit\transpiler\passes`", in their respective file path and the keyword "`run`" for the respective function name. Then, we extract the class names for each pass and the cumulative time spent in its `run` method and determine the 10 most costly passes with regard to overall runtime.

We categorize passes according to the module they are implemented in, e.g., passes in "`...\passes\synthesis`" as synthesis passes. To categorize passes in generic modules like `basis` or `utils`, we consider their stage of occurrence. For instance, *MinimumPoint*, which is from the `utils` module, occurs in the optimization stage; thus, we assign it to the circuit optimization category. We leave passes from generic modules that occur in multiple stages uncategorized.

Our preliminary analysis includes two quantum algorithms: the QFT [24] and the GHZ state [25] preparation. Both algorithms are implemented with 100 qubits to make sure that our test scenarios are representative of current workloads. We conduct all experiments using QPUs from the IBM platform to compare compilation times against actual quantum resource consumption. Specifically, we use the ibm_brisbane QPU, which is an Eagle r3 processor type. We perform a total of 30 executions per configuration (per circuit and optimization level). We executed quantum programs using Qiskit v1.3.2 and Python 3.9.2, and all stages of the Qiskit compiler pipeline are executed on an HPC node with a total number of 48 available CPU cores (Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz) with 250 GB of memory running Debian Linux 11.

[1]A profile is a set of statistics that describe how often and for how long different parts of the program are executed [29].

V. Results & Discussion

Fig. 3 shows a summary of the compilation and QPU execution time contributions to the total runtime for different optimization levels. Compilation time for the QFT circuit takes between 20% and 42% of the total runtime for optimization levels 0, 1, and 2. Interestingly, we observe a decreasing trend in the compilation time for the QFT circuit as we increase the optimization level from 0 to 2, while optimization level 3 increases it up to 47.4 seconds, accounting for 82% of the total runtime. Notably, the QPU execution time varies between 18.5 and 10.3 seconds, showing a -44% reduction for increasing the optimization level from 0 to 3. Conversely, the GHZ circuit compilation time increases with higher optimization levels, accounting for up to 95% of the total runtime for optimization level 3. In this case, we observed a reduction of the QPU execution time of about -27% when comparing optimization levels 0 (4.1s) and 3 (3s). However, this marginal tradeoff comes at the expense of substantial compilation time.

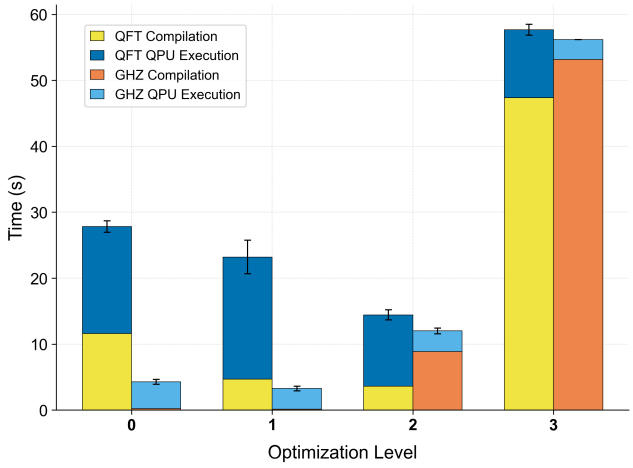
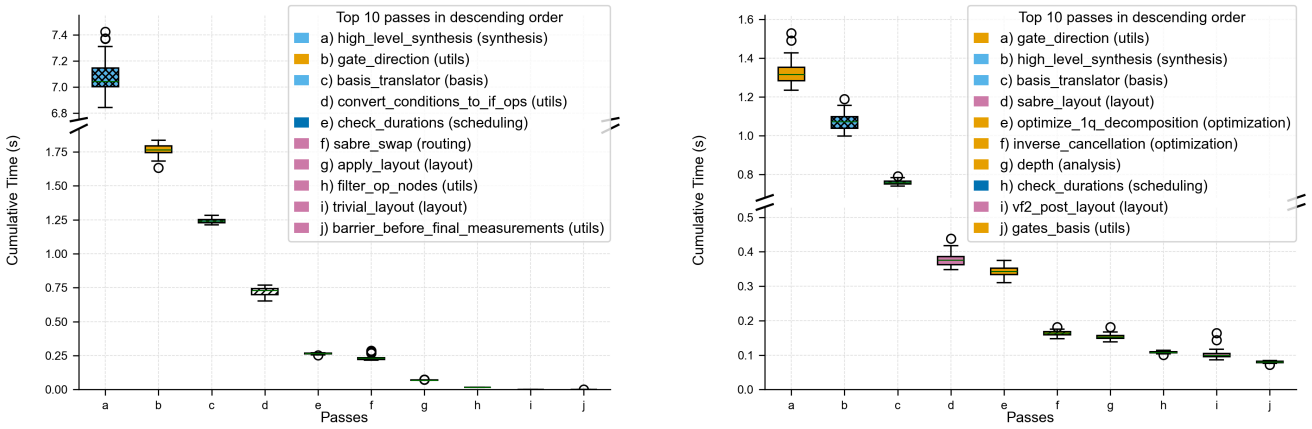


Fig. 3: Compilation and QPU execution times of the QFT and GHZ circuits for different optimization levels.

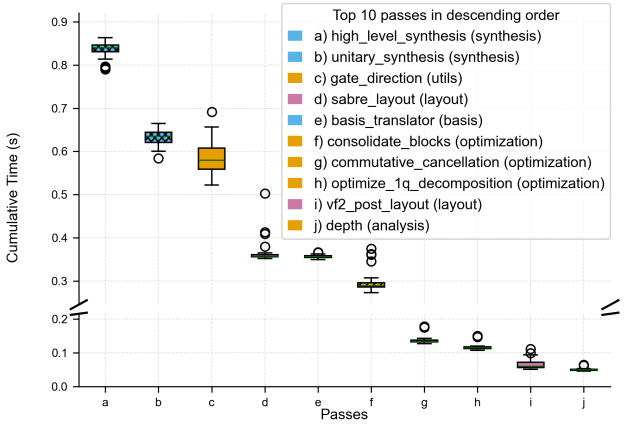
Figures 4 and 5 show the top 10 most costly compilation passes for the QFT and GHZ circuits, respectively. Both figures are divided into four sub-figures (a)–(d) for each optimization level, and each figure presents box plots of the cumulative time spent executing the corresponding passes.

The results for the QFT circuit (Fig. 4) show that among the top 10 costly passes, five are from the optimization category for optimization levels 1 and 2, and six for optimization level 3. For optimization level 0, a single pass from the same category is observed. Passes from the synthesis category represent two of the top 10 passes for optimization levels 0 and 1, and three of the top 10 for optimization levels 2 and 3. Mapping passes are represented by five out of the top 10 for optimization level 0, but with an increasing optimization level, they are replaced by passes from the optimization category.

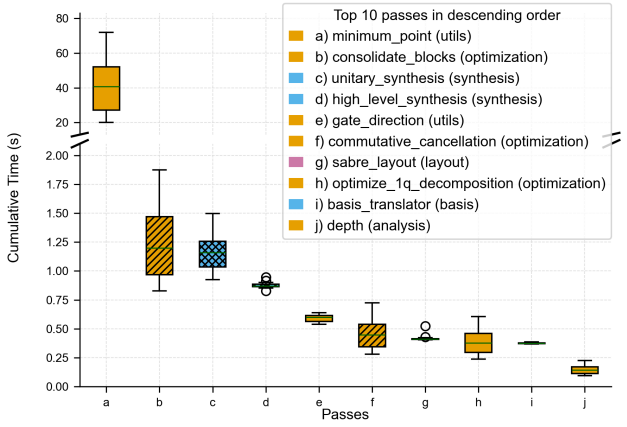


(a) Optimization Level 0

(b) Optimization Level 1



(c) Optimization Level 2



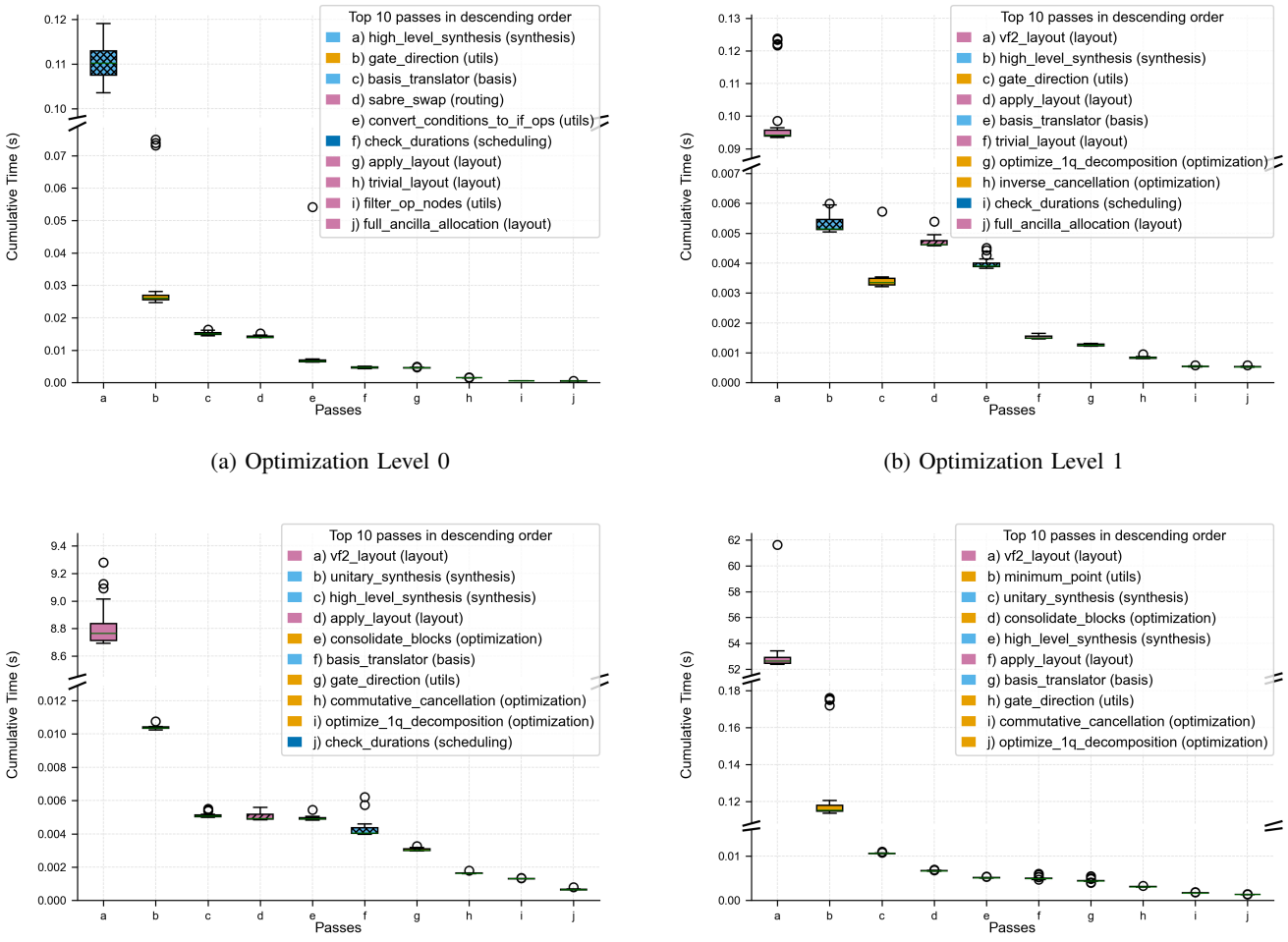
(d) Optimization Level 3

Fig. 4: Boxplots of the top 10 most expensive Qiskit preset compiler passes for the QFT circuit and optimization levels 0 (a), 1 (b), 2 (c), and 3 (d). Passes from the gate synthesis category are marked blue, qubit mapping passes are pink, and circuit optimization passes are orange. Scheduling passes are displayed in dark blue, while uncategorized passes remain white. Right-diagonal hatching (/) indicates passes that occur in two stages, and cross-hatching (x) indicates passes that occur in more than two stages.

The results for the GHZ state preparation circuit (Fig. 5) reveal a comparable trend to that observed in QFT for synthesis passes, with two among the top 10 for optimization levels 0 and 1, and three among the top 10 for optimization levels 2 and 3. The number of optimization passes among the top 10 also increases with respect to the optimization level, as is the case with QFT. Specifically, we observe one, three, four, and five passes for GHZ with optimization levels 0, 1, 2, and 3, respectively. Furthermore, mapping passes consume a substantial portion of the overall runtime in comparison to QFT. In particular, for the GHZ circuit and optimization levels 1, 2, and 3, the *VF2Layout* pass requires several orders of magnitude longer execution times (up to 61.6s) compared to all other passes. With optimization levels 2 and 3, this pass even accounts for over 99% of total compilation time.

A notable observation for both circuits is the significantly higher cumulative time for *HighLevelSynthesis* for optimization level 0, which is ≈21x higher in the case of GHZ (up to 0.168s) and ≈7x higher in the case of QFT (up to 7.43s) when compared to other optimization levels with the same circuit. Optimization level 0 does not perform high-level optimizations prior to QCC, which is why gate synthesis may require more time. However, it is also important to note that the runtime of synthesis passes is approximately 100x less for GHZ than for the same passes in QFT, which is likely due to the different circuit depths.

The *MinimumPoint* pass, which occurs only in optimization level 3, consumes a significant amount of time compared to other passes for both circuits, although its impact is stronger with QFT. It is worth noting that for QFT this procedure



(a) Optimization Level 0

(b) Optimization Level 1

(c) Optimization Level 2

(d) Optimization Level 3

Fig. 5: Boxplots of the top 10 most expensive Qiskit preset compiler passes for the GHZ state preparation and optimization levels 0 (a), 1 (b), 2 (c), and 3 (d). Passes from the gate synthesis category are marked blue, qubit mapping passes are pink, and circuit optimization passes are orange. Scheduling passes are displayed in dark blue, while uncategorized passes remain white. Right-diagonal hatching (/) indicates passes that occur in two stages, and cross-hatching (x) indicates passes that occur in more than two stages.

accounts for about 87% of total compilation time and can even take up to 72 seconds, the highest cumulative value for an individual pass across all executions. The data used for all figures and results in this work is available via Zenodo[2].

VI. Conclusions and Future Work

The primary contribution of this work is a preliminary analysis of the execution profile of the Qiskit built-in compiler toolchain, with the goal of examining the cumulative time spent on each individual pass. Our results represent the first step in our profiling approach to identifying costly passes, which helps researchers and developers detect potential bottlenecks in the compilation process and supports the effective use of precompilation techniques. Our current profiling pipeline measures the cumulative time of a given pass across all stages in which it may occur. As the built-in `PassManager` from Qiskit executes stages sequentially with no overlap between stages, stage-aware profiling could offer more profound insights into the contributions of each pass per stage. Currently, our analysis is SDK dependent, but similar examinations for other SDKs seem worthwhile. In the future, we aim to extend our analysis to a more extensive set of circuits along with stage-aware profiling and include the actual outcome of the quantum computation in our analysis.

[2]https://doi.org/10.5281/zenodo.15255700

Acknowledgments

This work is funded by the Internet Stiftung through the Netidee scholarship ID 7413 (Optimizing Hybrid Workflows for Cloud-Based Quantum Computation).

We acknowledge the use of IBM Quantum Credits for this work. The views expressed are those of the authors and do not reflect the official policy or position of IBM or the IBM Quantum team. In this paper, we used ibm_brisbane, which is of the IBM Quantum Eagle r3 processor type.

REFERENCES

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[2] A. Bayerstadler, G. Becquin, J. Binder, T. Botter, H. Ehm, T. Ehmer, M. Erdmann, N. Gaus, P. Harbach, M. Hess, J. Klepsch, M. Leib, S. Luber, A. Luckow, M. Mansky, W. Mauerer, F. Neukart, C. Niedermeier, L. Palackal, R. Pfeiffer, C. Polenz, J. Sepulveda, T. Sievers, B. Standen, M. Streif, T. Strohm, C. Utschig-Utschig, D. Volz, H. Weiss, and F. Winter, "Industry quantum computing applications," *EPJ Quantum Technology*, vol. 8, no. 1, Nov. 2021.

[3] B. Bauer, S. Bravyi, M. Motta, and G. K.-L. Chan, "Quantum algorithms for quantum chemistry and quantum materials science," *Chemical Reviews*, vol. 120, no. 22, p. 12685–12717, Oct. 2020.

[4] A. Abbas, A. Ambainis, B. Augustino, A. Bärtschi, H. Buhrman, C. Coffrin, G. Cortiana, V. Dunjko, D. J. Egger, B. G. Elmegreen, N. Franco, F. Fratini, B. Fuller, J. Gacon, C. Gonciulea, S. Gribling, S. Gupta, S. Hadfield, R. Heese, G. Kircher, T. Kleinert, T. Koch, G. Korpas, S. Lenk, J. Marecek, V. Markov, G. Mazzola, S. Mensa, N. Mohseni, G. Nannicini, C. O'Meara, E. P. Tapia, S. Pokutta, M. Proissl, P. Rebentrost, E. Sahin, B. C. B. Symons, S. Tornow, V. Valls, S. Woerner, M. L. Wolf-Bauwens, J. Yard, S. Yarkoni, D. Zechiel, S. Zhuk, and C. Zoufal, "Challenges and opportunities in quantum optimization," *Nature Reviews Physics*, vol. 6, no. 12, p. 718–735, Oct. 2024.

[5] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, p. 195–202, Sep. 2017.

[6] S. Gulde, M. Riebe, G. P. T. Lancaster, C. Becher, J. Eschner, H. Häffner, F. Schmidt-Kaler, I. L. Chuang, and R. Blatt, "Implementation of the deutsch–jozsa algorithm on an ion-trap quantum computer," *Nature*, vol. 421, no. 6918, p. 48–50, Jan. 2003.

[7] L. DiCarlo, J. M. Chow, J. M. Gambetta, L. S. Bishop, B. R. Johnson, D. I. Schuster, J. Majer, A. Blais, L. Frunzio, S. M. Girvin, and R. J. Schoelkopf, "Demonstration of two-qubit algorithms with a superconducting quantum processor," *Nature*, vol. 460, no. 7252, p. 240–244, Jun. 2009.

[8] N. Maring, A. Fyrillas, M. Pont, E. Ivanov, P. Stepanov, N. Margaria, W. Hease, A. Pishchagin, A. Lemaître, I. Sagnes, T. H. Au, S. Boissier, E. Bertasi, A. Baert, M. Valdivia, M. Billard, O. Acar, A. Brieussel, R. Mezher, S. C. Wein, A. Salavrakos, P. Sinnott, D. A. Fioretto, P.-E. Emeriau, N. Belabas, S. Mansfield, P. Senellart, J. Senellart, and N. Somaschi, "A versatile single-photon-based quantum computing platform," *Nature Photonics*, vol. 18, no. 6, p. 603–609, Mar. 2024.

[9] P. J. Karalekas, N. A. Tezak, E. C. Peterson, C. A. Ryan, M. P. da Silva, and R. S. Smith, "A quantum-classical cloud platform optimized for variational hybrid algorithms," *Quantum Science and Technology*, vol. 5, no. 2, p. 024003, Apr. 2020.

[10] T. Beck, A. Baroni, R. Bennink, G. Buchs, E. A. C. Pérez, M. Eisenbach, R. F. da Silva, M. G. Meena, K. Gottiparthi, P. Groszkowski, T. S. Humble, R. Landfield, K. Maheshwari, S. Oral, M. A. Sandoval, A. Shehata, I.-S. Suh, and C. Zimmer, "Integrating quantum computing resources into scientific hpc ecosystems," *Future Generation Computer Systems*, vol. 161, p. 11–25, Dec. 2024.

[11] M. Ruefenacht, B. Taketani, M. Weber, P. Lähteenmäki, V. Bergholm, D. Kranzlmüller, L. Schulz, and M. Schulz, "Bringing quantum acceleration to supercomputers," 05 2022.

[12] E. Pelofske, A. Bärtschi, L. Cincio, J. Golden, and S. Eidenbenz, "Scaling whole-chip qaoa for higher-order ising spin glass models on heavy-hex graphs," *npj Quantum Information*, vol. 10, no. 1, Nov. 2024.

[13] R. C. Farrell, M. Illa, A. N. Ciavarella, and M. J. Savage, "Scalable circuits for preparing ground states on digital quantum computers: The schwinger model vacuum on 100 qubits," *PRX Quantum*, vol. 5, no. 2, Apr. 2024.

[14] L. S. Madsen, F. Laudenbach, M. F. Askarani, F. Rortais, T. Vincent, J. F. F. Bulmer, F. M. Miatto, L. Neuhaus, L. G. Helt, M. J. Collins, A. E. Lita, T. Gerrits, S. W. Nam, V. D. Vaidya, M. Menotti, I. Dhand, Z. Vernon, N. Quesada, and J. Lavoie, "Quantum computational advantage with a programmable photonic processor," *Nature*, vol. 606, no. 7912, p. 75–81, Jun. 2022.

[15] Y. Wu, W.-S. Bao, S. Cao, F. Chen, M.-C. Chen, X. Chen, T.-H. Chung, H. Deng, Y. Du, D. Fan, M. Gong, C. Guo, C. Guo, S. Guo, L. Han, L. Hong, H.-L. Huang, Y.-H. Huo, L. Li, N. Li, S. Li, Y. Li, F. Liang, C. Lin, J. Lin, H. Qian, D. Qiao, H. Rong, H. Su, L. Sun, L. Wang, S. Wang, D. Wu, Y. Xu, K. Yan, W. Yang, Y. Yang, Y. Ye, J. Yin, C. Ying, J. Yu, C. Zha, C. Zhang, H. Zhang, K. Zhang, Y. Zhang, H. Zhao, Y. Zhao, L. Zhou, Q. Zhu, C.-Y. Lu, C.-Z. Peng, X. Zhu, and J.-W. Pan, "Strong quantum computational advantage using a superconducting quantum processor," *Physical Review Letters*, vol. 127, no. 18, Oct. 2021.

[16] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with qiskit," 2024.

[17] N. Quetschlich, L. Burgholzer, and R. Wille, "Mqt predictor: Automatic device selection with device-specific circuit compilation for quantum computing," *ACM Transactions on Quantum Computing*, vol. 6, no. 1, p. 1–26, Jan. 2025.

[18] F. Leymann, J. Barzen, M. Falkenthal, D. Vietz, B. Weder, and K. Wild, "Quantum in the cloud: Application potentials and research opportunities," in *Proceedings of the 10th International Conference on Cloud Computing and Services Science*. SCITEPRESS - Science and Technology Publications, 2020.

[19] N. Quetschlich, L. Burgholzer, and R. Wille, "Reducing the compilation time of quantum circuits using pre-compilation on the gate level," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 01, 2023, pp. 757–767.

[20] "Qaching - documentation." [Online]. Available: https://qrisp.eu/reference/Jasp/qache.html

[21] J. Cheng, H. Deng, and X. Qia, "Accqoc: Accelerating quantum optimal control based pulse generation," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 543–555.

[22] P. Gokhale, Y. Ding, T. Propson, C. Winkler, N. Leung, Y. Shi, D. I. Schuster, H. Hoffmann, and F. T. Chong, "Partial compilation of variational algorithms for noisy intermediate-scale quantum machines," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. ACM, Oct. 2019.

[23] D. Kudrow, K. Bier, Z. Deng, D. Franklin, Y. Tomita, K. R. Brown, and F. T. Chong, "Quantum rotations: a case study in static and dynamic machine-code generation for quantum computers," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA'13. ACM, Jun. 2013, p. 166–176.

[24] S. Pattanayak, *Quantum Fourier Transform and Related Algorithms*. Apress, 2021, p. 151–220.

[25] D. M. Greenberger, M. A. Horne, and A. Zeilinger, *Going Beyond Bell's Theorem*. Springer Netherlands, 1989, p. 69–72.

[26] A. Botea, A. Kishimoto, and R. Marinescu, "On the complexity of quantum circuit compilation," *Proceedings of the International Symposium on Combinatorial Search*, vol. 9, no. 1, p. 138–142, Sep. 2021.

[27] [Online]. Available: https://docs.quantum.ibm.com/api/qiskit/transpiler

[28] R. Seidel, S. Bock, R. Zander, M. Petrič, N. Steinmann, N. Tcholtchev, and M. Hauswirth, "Qrisp: A framework for compilable high-level programming of gate-based quantum computers," 2024.

[29] Python Software Foundation, "The python profilers," 2025, accessed on: 26.03.2025. [Online]. Available: https://docs.python.org/3/library/profile.html

[30] "2023 quantum open source survey." [Online]. Available: https://unitaryfund.github.io/survey-website/

[31] P. D. Nation, A. A. Saki, S. Brandhofer, L. Bello, S. Garion, M. Treinish, and A. Javadi-Abhari, "Benchmarking the performance of quantum computing software for quantum circuit creation, manipulation and compilation," *Nature Computational Science*, Apr. 2025.