Article

An Extensible Computing Platform for Heavy Quantum Simulation Workloads

Andrés García [1], José Ranilla [2], Raúl Alonso [3] and Luis Meijueiro [4,*]

[1] CTIC Centro Tecnológico, Parque Científico Tecnológico de Gijón, Ada Byron 39, 33203 Gijón, Spain; andres.garcia@fundacionctic.org
[2] Computer Science Department, Escuela Politécnica de Ingeniería de Gijón, Universidad de Oviedo, Pedro Puig Adam s/n, 33204 Gijón, Spain; ranilla@uniovi.es
[3] CTIC Centro Tecnológico, Parque Científico Tecnológico de Gijón, Ada Byron 39, 33203 Gijón, Spain; raul.alonso@fundacionctic.org
[4] CTIC Centro Tecnológico, Parque Científico Tecnológico de Gijón, Ada Byron 39, 33203 Gijón, Spain; luis.meijueiro@fundacionctic.org
[*] Correspondence: luis.meijueiro@fundacionctic.org

Abstract: The shortage of quantum computers, and their current state of development, constraints research in many fields that could benefit from quantum computing. Although the work of a quantum computer can be simulated with classical computing, personal computers take so long to run quantum experiments that they are not very useful for the progress of research. This manuscript presents an open quantum computing simulation platform that enables quantum computing researchers to have access to high performance simulations. This platform, called QUTE, relies on a supercomputer powerful enough to simulate general purpose quantum circuits of up to 38 qubits, and even more under particular simulations. This manuscript describes in-depth the characteristics of the QUTE platform and the results achieved in certain classical experiments in this field, which would give readers an accurate idea of the system capabilities.

Keywords: quantum computing; quantum computing framework; quantum simulator; quantum technology; quantum theory; quantum mechanics; quantum circuit; quantum algorithm; ibm qiskit; notebooks jupyter

1. Introduction

In recent years, the quantum computing paradigm has become increasingly popular in the scientific and technological community at large. While a classical computer operates with binary digits or bits, a quantum computer uses quantum bits, or qubits, which are subject to the principles of quantum mechanics, such as superposition and entanglement [1]. These peculiar properties enable quantum computing to provide an alternative way of approaching computationally hard problems, offering an asymptotic speedup for certain tasks, including factoring [2] -with dramatic consequences for widely used cryptographic protocols such as RSA- and searching in unstructured databases [3].

One of the main difficulties that arises when trying to exploit the advantages of quantum computing techniques is the limitation of current quantum hardware due to imperfections, noise and restricted scalability. Quantum hardware still seems to be a few years away from being able to solve practical problems faster than traditional computers. In addition to their current state of development, there is a shortage of quantum computers, constraining research in many fields that could benefit from quantum computing.

For these reasons, the use of classical simulators of quantum devices is of paramount importance for implementing, testing and tuning quantum solutions. This manuscript presents an open quantum computing simulation platform, called QUTE, that allows quantum computing researchers to have access to high performance simulations.

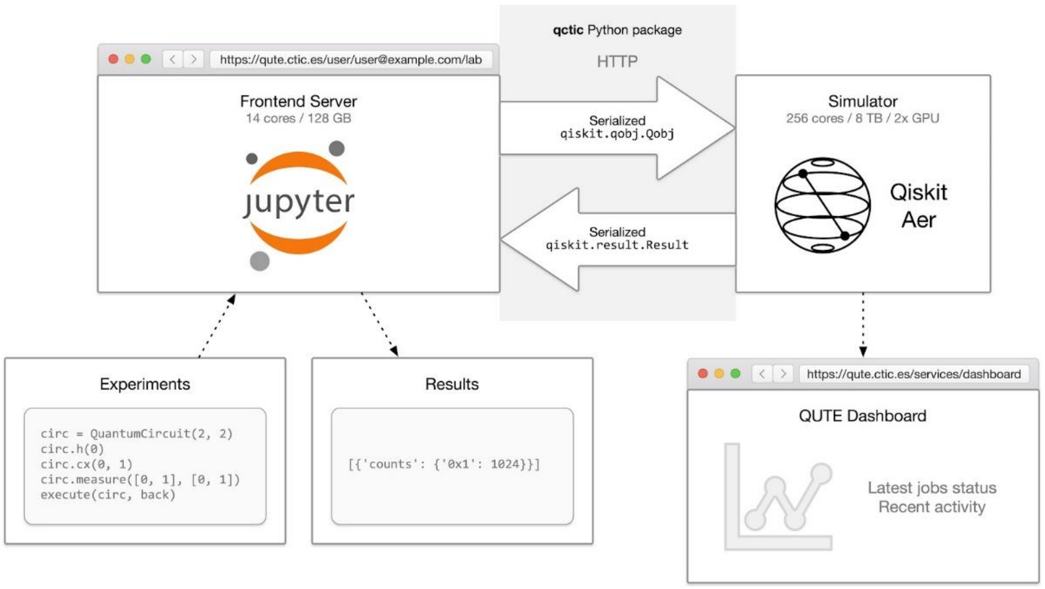
The platform is installed in a supercomputer powerful enough to simulate general purpose quantum circuits of up to 38 qubits, and even more under particular simulations. The frontend of QUTE is based on an isolated Jupyter Notebook [4] for each user. This environment includes the most common Python modules and the latest Qiskit stack [5]. It means that any existing vanilla Qiskit Python script should run out of the box on the frontend. In order for the notebook script to take full advantage of the simulator capabilities, a Qiskit compatible API has been developed. Since many researchers can be using the platform at the same time, a Job Manager receives and enqueues the requests sent via QCTIC API. The Job manager then makes sure that the experiments are run in the most efficient fashion to take full advantage of the hardware capabilities of the simulator.

2. Materials and Methods

This section begins with a description of the QUTE architecture, including all the distinct components that can be identified in the simulation platform, as well as their interactions. The subsequent part entitled Job Lifecycle provides details of the overall lifecycle process every job goes through. Finally, the Experiments subsection presents the experiments that we have conducted in order to test the performance of the QUTE simulator.

2.1. QUTE Architecture

Figure 1 below provides an overview of these components and shows the typical workflow for experiments in QUTE.


qctic Python package
HTTP
https://qute.ctic.es/user/user@example.com/lab
Frontend Server
14 cores / 128 GB
jupyter
Serialized qiskit.qobj.Qobj
Serialized qiskit.result.Result
Simulator
256 cores / 8 TB / 2x GPU
Qiskit Aer
Experiments
Results
https://qute.ctic.es/services/dashboard
QUTE Dashboard
Latest jobs status
Recent activity

```
circ = QuantumCircuit(2, 2)
circ.h(0)
circ.cx(0, 1)
circ.measure([0, 1], [0, 1])
execute(circ, back)
```

```
[{'counts': {'0x1': 1024}}]
```

Figure 1. QUTE Components and workflow.

The backend simulator installed on the QUTE high-performance computing platform is the component in charge of executing the simulation jobs, serving as the foundation of the entire system. To that end, the computing platform is equipped with a CPU formed by 256 Intel® Xeon cores (Intel Corporation, Mountain View, CA, USA), 8TB of RAM memory, and has also two GPU NVIDIA Tesla V100 cards (NVIDIA Corporation, Santa Clara, CA, USA) with 16GB of on-board memory each. The simulator leverages the capabilities of the Docker containerisation platform [6] in order to easily create isolated and ephemeral execution environments for the simulation jobs.

The frontend server is the entry point to the platform for end users. It is based on JupyterLab, an application that provides extended functionality to interface with Jupyter notebooks using the browser. Jupyter notebooks, in turn, are a highly relevant tool in the scientific community that allows for the combination of interactive code, modern visualisations and documentation in a single view. All QUTE users have their own isolated JupyterLab instance that is already provisioned with all the necessary requirements to interface with the simulator, including software packages and environment configuration.

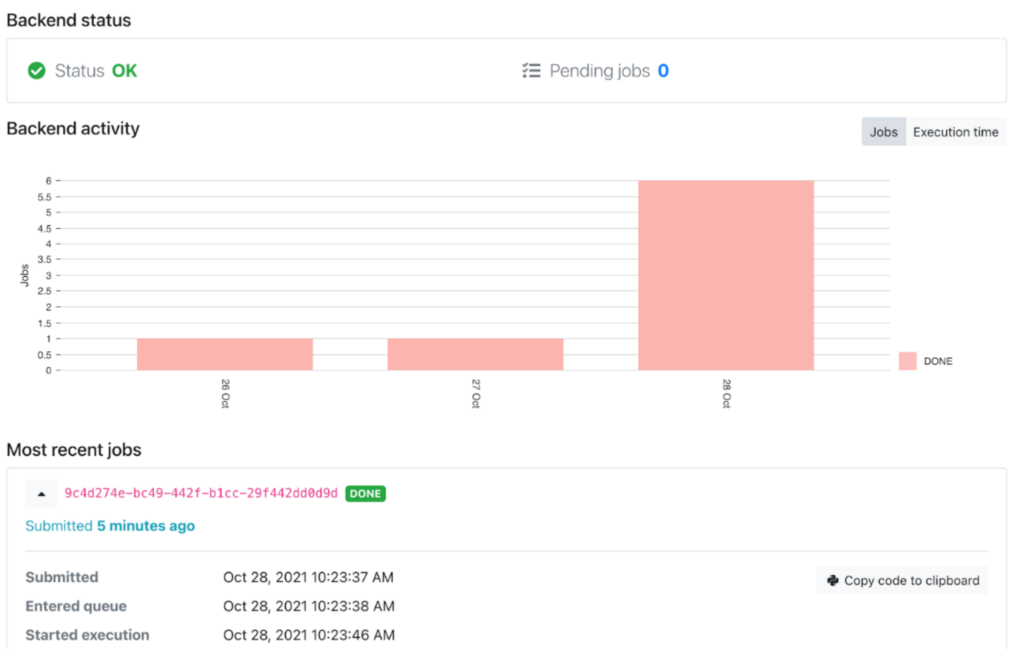
Users may run experiments from the frontend server by using the QCTIC package [7]. QCTIC is a Python package that mirrors the interface of the Aer Qiskit Provider with the aim of being transparent and familiar to users with previous Qiskit experience. There are two relevant differences between QCTIC and Aer:

- QCTIC does not contain any simulation logic, that is, all Aer simulation jobs are passed to the backend simulator component using the QCTIC HTTP API. The main purpose of QCTIC is to handle communications between the frontend server and the backend simulator, serialising the relevant objects and abstracting the user from the low-level details of the communications between these two entities;
- QCTIC contributes to the Qiskit API by introducing a complementary interface based on the asynchronous I/O programming model of asyncio (a library built into Python 3 that enables developers to write concurrent code using the async/await syntax). This single-threaded model uses asynchronous coroutines instead of callbacks or blocking statements and is especially suited for programs with a heavy component of network requests, as is the case with QCTIC. It is important to note that this is an optional interface, and the regular blocking Qiskit interface is still available for users with that preference, which enables them to run previously existing programs with minor or no modifications.

As mentioned previously, the backend simulator exposes an HTTP API that serves as the programmatic interface to the Qiskit Aer simulator installed in the high-performance computing platform. The API follows the REST architectural model; although not in a strict fashion (i.e. it does not fulfil the HATEOAS requirement of guiding the client along with the responses provided). It is fairly simple and consists of two entities:

- A Job represents a single experiment and is composed of a series of timestamps, a unique job identifier, the current status of the job, a serialised Qiskit Qobj object that is a self-contained representation of the experiment, and a serialised Qiskit Result object if the experiment finished successfully, or an error object otherwise. Jobs are the main unit of information exchanged by the distinct components of the QUTE platform;
- A BackendStatus instance represents the current state of the backend simulator, including the number of jobs that are waiting in the queue to be executed and a flag indicating if the service is operational.

Finally, the QUTE Dashboard is a web application that is separate from the JupyterLab service and provides a view of the state of the QUTE service from the point of view of the user. This includes the latest jobs for which the user is the owner, and high-level statistics of the usage of the computational resources. Please, see Figure 2 below for a screenshot of this dashboard.


Backend status

Status OK

Pending jobs 0

Backend activity

Most recent jobs

```
9c4d274e-bc49-442f-b1cc-29f442dd0d9d DONE
```

Submitted 5 minutes ago

Submitted	Oct 28, 2021 10:23:37 AM
Entered queue	Oct 28, 2021 10:23:38 AM
Started execution	Oct 28, 2021 10:23:46 AM

Figure 2. Example of QUTE Dashboard interface.

2.2. *Job Lifecycle*

QUTE simulation jobs use the same life cycle model as Qiskit, that is, the possible statuses of QUTE jobs at any given stage are the same as those defined by the JobStatus enumeration in Qiskit. Firstly, the user writes the experiment on top of Qiskit using the QCTIC provider—this provider exposes a proxy backend for each one of the backends included in Aer. A Job will be initialised for each distinct Qiskit execution (i.e. call to the execute function).

Then, jobs will be appended to the execution queue in a strict FIFO fashion. It should be noted that currently there is no parallelism, that is, only one job can be executed at any given time. The rationale behind this design decision is that one of the main contributions of the QUTE platform is the ability to run quantum simulation experiments with a number of qubits that far exceeds the capabilities of a regular workstation, thus, an experiment needs to be able to utilise the full capabilities of the hardware. Anyway, further optimisations towards enabling higher levels of parallelism would be possible, especially in the context of distributing workloads between the GPUs and the CPUs.

Jobs are executed in an isolated environment by leveraging the Docker platform and the NVIDIA container toolkit, which is a requirement to enable NVIDIA GPU support. When jobs are created, they first go through an initialisation process (i.e. INITIALIZING stage) before being enqueued in the job queue and updated to the QUEUED state. Jobs can then evolve to one of two states: if a job finishes successfully, its serialised Qiskit result will be persisted in the data store and its status will be updated to DONE; if a job fails, the exception will be logged and the status will be updated to ERROR. In both cases, a notification will be pushed to the user, which will then have the option to retrieve the job instance using its unique ID for further processing. Please see Figure 3 for a diagram representing a simplified view of the life cycle of a job.

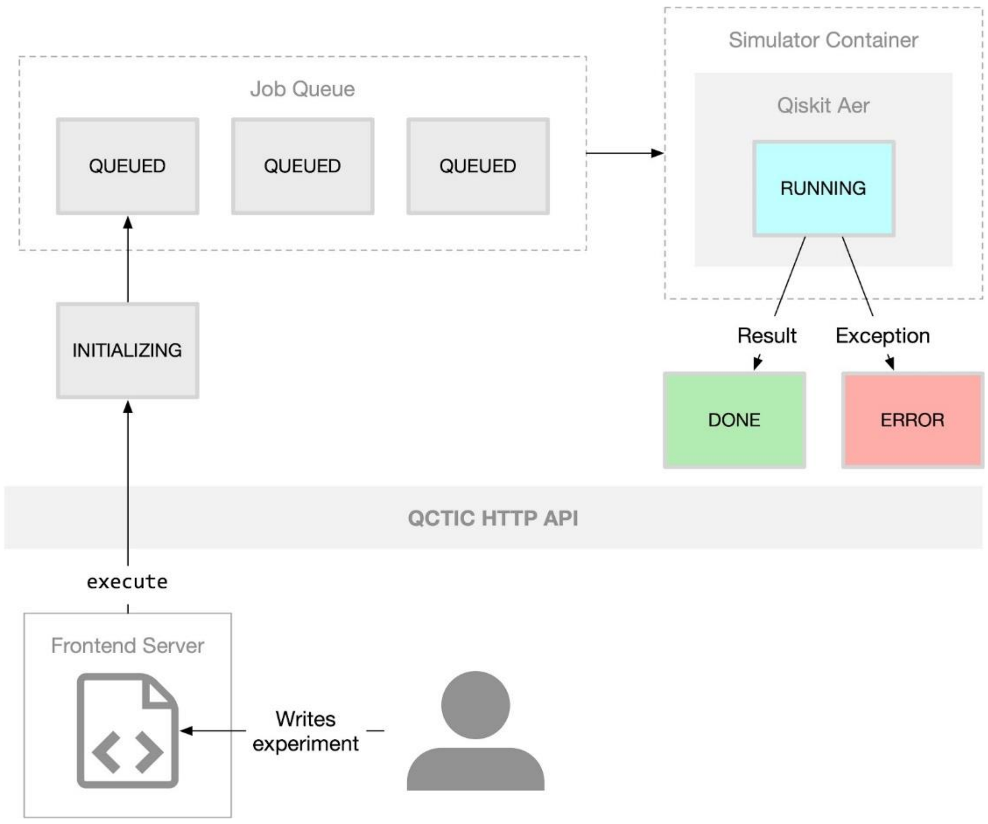


Figure 3. QUTE Job Life Cycle.

2.3. Experiments

In this section are presented the settings of the experiments we have conducted in order to test the performance of the QUTE simulator.

The experiments were designed in such a way as to compare our system to other online available Qiskit-based simulators, as well as to other high-performance computing clusters not specifically designed for quantum circuit simulation.

For the experiments, several systems to test against QUTE were selected:

- The first one is the quantum simulator provided as a part of IBM Quantum [8] (IBM Corporation, Armonk, NY, USA) which supports up to 32 qubits (to the best of our knowledge, this is the highest number of qubits of any quantum simulator publicly available online). The details of the simulator hardware architecture are not known, but it runs Qiskit out-of-the-box;
- As another online, publicly available alternative, we have considered Google Colab [9] (Google, Mountain View, CA, USA), especially because it allows the use of GPUs, something not provided by the IBM Quantum simulator;
- In addition, we have used local clusters located at the University of Oviedo Computer Science Department [10]. The first one has 20 cores at 2.2 Ghz and 256 MB of RAM. The second one has 16 cores at 1.6 Ghz and 1TB of RAM, as well as a NVIDIA Tesla P100 GPU.

The benchmarks have been run on random quantum circuits of 20 to 38 qubits (or the maximum number of qubits supported by each simulator) with a number of layers ranging from 10 to 50. The circuits were generated using Qiskit's function random_circuit (Figure 4 shows an example of a circuit of 5 qubits and 10 layers generated with that method) and the same seed was used in all the systems to guarantee a fair comparison. We generate 50 circuits of each size and number of layers (for the highest number of qubits we reduced this to 10, or even 5 circuits, in order to obtain reasonable execution times) and compute the average and standard deviation of their running times. Throughout all the experiments, we use Qiskit version 0.25.3.

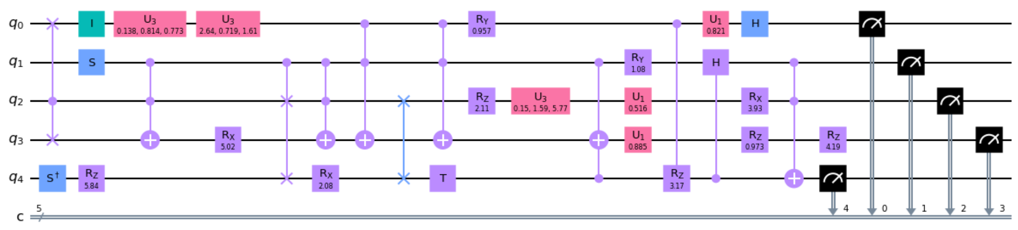


Figure 4. Example of a 5-qubit quantum circuit with 10 layers.

The choice of random circuits for testing the performance of the simulators follows the common practice of using this type of circuits for benchmarking quantum processors (see [11,12]). The simulation of random circuits is considered to be computationally hard [13] and is the base of the famous quantum supremacy experiments [14,15].

Notice, however, that since we are running the experiments on quantum simulators, we are not interested in computing fidelities [16] (which will be 1, by definition) but only on studying the way in which the running times scale and compare to each other in the different systems considered in this paper.

3. Results

In this section, the results obtained after executing the set of experiments are presented. Finally, we discuss its significance.

Figure 5 below shows the average running time of random circuits of 20 to 38 qubits with 10 layers. As expected, the scaling of the running time is exponential in the number of qubits (notice that the vertical axis is in logarithmic scale). The only system that supported circuits with more than 32 qubits was QUTE, which also consistently showed the lowest running times from 26 qubits onwards, with only small differences to the best system for 20, 22 and 24 qubits.

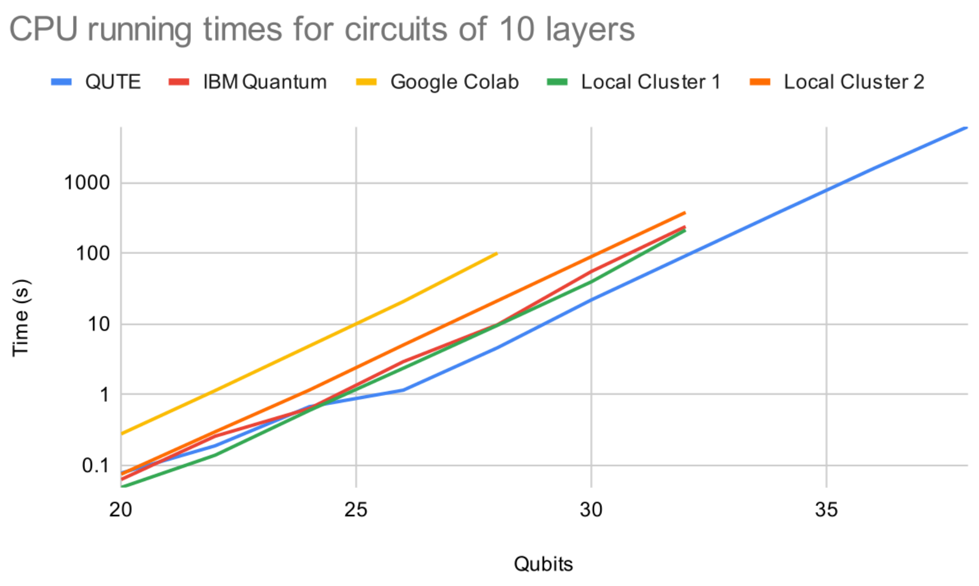


Figure 5. CPU running times for circuits of 10 layers.

The detailed results for circuits with 10, 20, 30, 40 and 50 layers when run on CPUs are shown in Table 1. Notice that, again, the average time grows exponentially with the number of qubits but only linearly with the number of layers. This is consistent with what is theoretically expected (see [13]).

Table 1. QUTE results (in seconds) when using CPUs.

	10 Layers		20 Layers		30 Layers		40 Layers		50 Layers	
Qubits	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std
20	0.08	0.01	0.14	0.01	0.20	0.01	0.26	0.01	0.32	0.01
22	0.19	0.02	0.36	0.03	0.53	0.04	0.70	0.04	0.87	0.05
24	0.68	0.07	1.29	0.10	1.89	0.14	2.51	0.15	3.12	0.16
26	1.16	0.07	2.03	0.11	2.83	0.12	3.59	0.25	4.54	0.21
28	4.61	0.25	8.11	0.45	11.78	1.06	15.30	1.07	18.86	1.51
30	21.94	1.20	39.42	1.44	54.71	1.91	71.19	2.67	89.41	3.65
32	91.91	6.03	163.67	6.22	232.21	8.98	307.14	10.02	378.17	14.47
34	384.32	33.72	678.66	30.82	962.07	44.51	1260.94	59.49	1553.28	56.41
36	1579.48	110.88	2674.39	41.10	3915.18	139.16				
38	6151.70	407.82	11080.19	452.63	16184.75	201.01				

Figure 6 presents the average running times of random circuits when using GPUs for the simulation. Two of the systems (IBM Quantum and Local Cluster 1) do not have GPUs and, consequently, were not considered for this set of experiments. On the other hand, Google Colab (randomly) assigns different GPUs when the user starts a session, so we present the results with two NVIDIA GPU models: the Tesla T4 and the Tesla K80.

Notice that these average running times (again, shown in logarithmic scale) are lower than those obtained when using CPUs, but the limited amount of RAM available on these devices prevents us from running circuits with more than 28 qubits. QUTE shows, again, the lowest running times across all the systems under study.

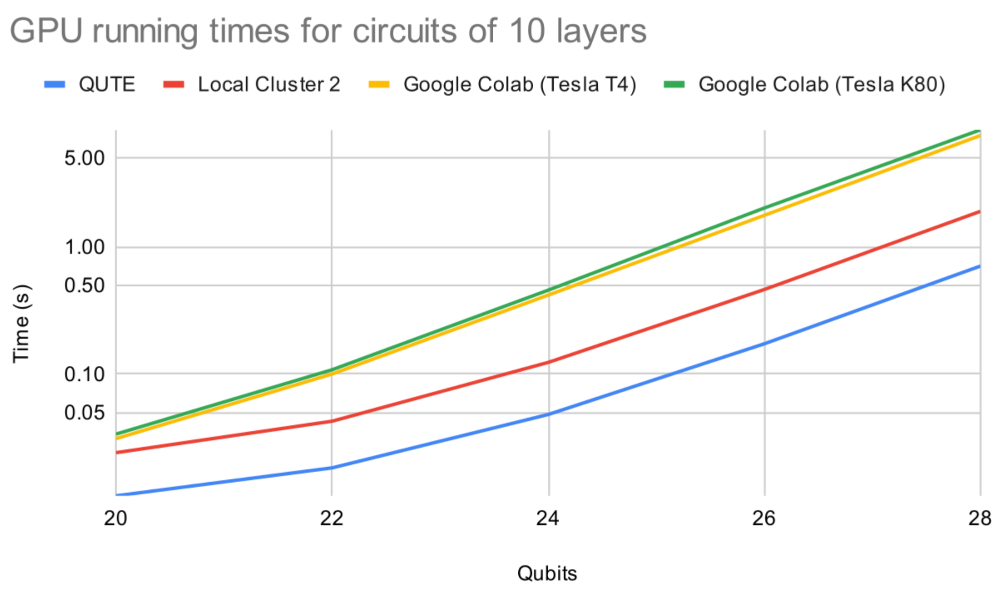


Figure 6. GPU running times for circuits of 10 layers.

Table 2 shows the detailed results of the experiments on GPUs. The trend is similar to the one found when using CPUs: exponential growth with the number of qubits and linear increase in time with the number of layers.

Notice that not only are the average times much lower than in the CPU case (whenever it is possible to run the circuit in the GPU), but also the standard deviations are much smaller.

Table 2. QUTE results (in seconds) when using GPUs.

	10 Layers		20 Layers		30 Layers		40 Layers		50 Layers	
Qubits	**Avg**	**Std**	**Avg**	**Std**	**Avg**	**Std**	**Avg**	**Std**	**Avg**	**Std**
20	0.01	0.00	0.02	0.00	0.02	0.00	0.02	0.00	0.03	0.00
22	0.02	0.00	0.03	0.00	0.04	0.00	0.05	0.00	0.06	0.00
24	0.05	0.00	0.09	0.00	0.12	0.00	0.16	0.00	0.20	0.00
26	0.17	0.01	0.33	0.01	0.48	0.01	0.64	0.01	0.80	0.02
28	0.71	0.03	1.37	0.04	2.04	0.05	2.70	0.06	3.36	0.07

For circuits of up to 28 qubits, where using GPUs is the fastest method, QUTE clearly outperforms all the other systems. For 30 and 32 qubits, when using CPUs, QUTE is again the fastest system (in fact, it is the fastest from 26 qubits, and for smaller circuits the differences are negligible). Moreover, we were only able to simulate circuits with 34 or more qubits by using QUTE, so it is the system of choice for the biggest circuits among those considered in these experiments.

The performance of the system scales as expected when increasing the number of qubits (roughly a factor of 2 when a new qubit is added) and when increasing the number of layers (with a linear increment in time when adding new layers).

4. Conclusions

QUTE shows the best overall performance among the systems under study throughout all the experiments we have run. It is also worth mentioning that QUTE seems to be the most stable of all the systems, with a standard deviation that is, in general, smaller than that obtained with the other simulators.

The successful development of the QUTE platform showed that there was still room for improvement in the performance of classic systems used for quantum computing simulation. A thoughtful design of the architecture and the optimisation of its components expand the real possibilities of using quantum computing simulators among the scientific community.

Furthermore, the architecture of the platform opens the possibility to future extensions for other quantum computing frameworks (e.g. ProjectQ [17]) while reusing a significant portion of the basic building blocks.

In our view, the adoption of QUTE platform could be very beneficial for researchers, as it provides the following advantages:

- Makes it possible to run quantum simulation experiments with high-qubit-count circuits, far exceeding the quantum simulation capabilities of high-end desktop computers, and even other current cloud-based simulation platforms;
- It gives the user a secure and private working environment, with the necessary tools to design from scratch or import their quantum experiments, as well as monitoring their execution;
- The modular design of the system allows for multiple configurations that facilitate the extension and optimisation of the system according to the current and future needs of the research community who use the platform.

Finally, we would like to remark that when these lines are written the use of QUTE platform is spreading among the scientific community, and the first results that make use of this platform are being published [18].

Author Contributions: Conceptualisation, A.G., J.R., R.A.; methodology J.R.; hardware and software, A.G., R.A.; writing, A.G., J.R., L.M.; supervision, L.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The authors confirm that the data is contained within the article or supplementary material.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Nielsen, M. A., & Chuang, I. (2002). Quantum computation and quantum information. American Journal of Physics 70, 558-559 (2002) https://doi.org/10.1119/1.1463744
2. P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. Proc. 35th Annual Symposium on Foundations of Computer Science, 1994. https://doi.org/10.1109/SFCS.1994.365700
3. L.K. Grover. A fast quantum mechanical algorithm for database search. https://doi.org/10.1145/237814.237866
4. Project Jupyter. Available online: https://jupyter.org (accessed on 25 October 2021)
5. Qiskit open source SDK. Available online: https://qiskit.org URL (accessed on 25 October 2021)
6. Docker. Available online: https://www.docker.com (accessed on 25 October 2021)
7. QCTIC Qiskit Provider. Available online: https://pypi.org/project/qctic/ (accessed on 25 October 2021)
8. IBM Quantum. Available online: https://quantum-computing.ibm.com (accessed on 25 October 2021)
9. Google Colab. Available online: https://colab.research.google.com (accessed on 25 October 2021)
10. University of Oviedo - Computer Sciences Available online: https://www.uniovi.es/en/departamentos/di (accessed on 25 October 2021)
11. Liu Y, Otten M, Bassirianjahromi R, Jiang L, Fefferman B. Benchmarking near-term quantum computers via random circuit sampling. arXiv preprint arXiv:2105.05232. 2021
12. Wootton JR. Benchmarking of quantum processors with random circuits. arXiv preprint arXiv:1806.02736. 2018
13. Boixo S, Isakov SV, Smelyanskiy VN, Babbush R, Ding N, Jiang Z, Bremner MJ, Martinis JM, Neven H. Characterizing quantum supremacy in near-term devices. Nature Physics. 2018 Jun;14(6):595-600.
14. Arute F, Arya K, Babbush R, Bacon D, Bardin JC, Barends R, Biswas R, Boixo S, Brandao FG, Buell DA, Burkett B. Quantum supremacy using a programmable superconducting processor. Nature. 2019 Oct;574(7779):505-10.
15. Wu Y, Bao WS, Cao S, Chen F, Chen MC, Chen X, Chung TH, Deng H, Du Y, Fan D, Gong M. Strong quantum computational advantage using a superconducting quantum processor. arXiv preprint arXiv:2106.14734. 2021
16. Wilde, Mark M. Quantum information theory. Cambridge University Press, 2013. https://doi.org/10.1017/CBO9781139525343
17. ProjectQ, an open-source software framework for quantum computing. Available online: https://projectq.ch/ (accessed on 25 October 2021)
18. Selomit R., Andrés E. R., Germán R., German S., Luiz V. Quantum algorithm for Feynman loop integrals. arXiv:2105.08703 2021