Investigating parallel execution of quantum Machine Learning circuits on superconducting hardware

Julien Rauch, Brice Chichereau, Stéphane Vialle, Patrick Carribault, Damien Rontani

Investigating parallel execution of quantum Machine Learning circuits on superconducting hardware

1st Julien Rauch
LISN UMR-9015, CNRS,
Université Paris-Saclay
F-91405 Orsay, France
julien.rauch@lisn.fr
0009-0009-4778-7692

2nd Brice Chichereau
CEA, DAM, DIF
F-91297 Arpajon, France
LIHPC, Université Paris-Saclay, CEA
Bruyères-le-Châtel, France
brice.chichereau@cea.fr

3rd Stéphane Vialle
LISN UMR-9015, CNRS,
Université Paris-Saclay,
CentraleSupélec
F-91405 Orsay, France
stephane.vialle@centralesupelec.fr
0000-0001-6336-2269

4th Patrick Carribault
CEA, DAM, DIF
F-91297 Arpajon, France
LIHPC, Université Paris-Saclay, CEA
Bruyères-le-Châtel, France
patrick.carribault@cea.fr

5th Damien Rontani
LMOPS EA-4423, Université de Lorraine,
CentraleSupélec
F-57070 Metz, France
Chair in Photonics, CentraleSupélec, Metz, France
damien.rontani@centralesupelec.fr
0000-0002-8549-4040

***Abstract*—Quantum Machine Learning algorithms generally rely on hybrid implementation with a classical learning loop running on a CPU and small quantum circuits running on a Quantum Processing Unit (QPU) for serialized processing batches of data. Considering the scarcity of quantum resources, parallelizing the execution of multiple instances of small quantum circuits instead of allocating a single circuit would better use quantum hardware resources. However, exploiting all available qubits in QPUs could result in a noisier operating condition and disrupt or slow the learning mechanism.**

This paper investigates the parallelization of QML algorithms on QPU for data clustering based on trainable generative models. We design a parallel macro-circuit with the Qiskit framework and measure their performance on an IBM superconducting quantum computer. A theoretical performance model is then proposed to determine the expected level of acceleration. Finally, we measure the impact of the QPU's occupancy on the loss functions of our QML algorithm, enabling us to identify the most reliable and exploitable acceleration range.

***Index Terms*—Parallel Quantum Circuits, Qiskit, Experimental Performance, Speedup, Quantum Machine Learning**

I. Motivations and objectives

Today's QPUs are available, but they remain rare and are shared by many users and many applications. Of course, the number of QPUs is increasing, but so is their user community. Execution time on a QPU is therefore precious, and this situation is set to continue for some years yet. Access to a QPU in the cloud is quite expensive once you exceed a small quota of free access time. It is therefore advisable not to waste runtime on a QPU.

This work was supported by the French government under the France 2030 program (QuanTEdu-France) under reference ANR-22-CMAS-0001.
The authors acknowledge also the financial support of the Chair in Photonics and the Région Grand-Est, France.

On the other hand, QPU manufacturers are increasing the size of QPUs to support more qubits and deeper circuits, improving QPU connectivity to facilitate the entanglement of any qubits, and attempting to reduce quantum noise inside QPUs. But many machine learning (ML) algorithms are iterative, and must apply the same calculations to different subsets of data. At each iteration, they have to run small quantum circuits of fixed depth several times on independent sets of qubits. They must then take measurements at the output of the QPU and send them back to the CPU to decide on the nature of the next iteration.

It therefore seems interesting to load onto the QPU a *macro-circuit* containing several independent small circuits, or even k times the same small circuit, until all the available qubits are used. So, instead of executing N times a small circuit, we would only execute N/k times a macro-circuit comprising k small circuits. However the following issues must be investigated:

- Will the execution time of an iteration on the QPU be divided by k (leading to an acceleration of k on the QPU)?
- Will quantum noise increase if more qubits are used to simultaneously execute more small independent circuits, disrupting the convergence of the ML algorithm and leading to more iterations being executed?
- Is there an optimal macro-circuit size to reduce the total execution time on the QPU (for the whole learning algorithm)?

This paper summarizes our design of macro-circuits simultaneously executing many small circuits, in order to accelerate a data clustering algorithm on a hybrid CPU-QPU architecture.

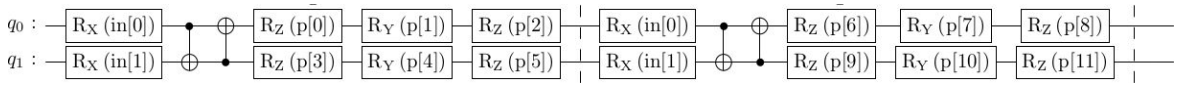

Fig. 1. Continuous QCBM circuit for 2 dimension data

We used generative networks, the Qiskit+Pytorch development environment and conducted our experiments both with Qiskit simulations and on real IBM quantum machines.

II. Related work and positioning

A. Benchmarking methodology

In [1], the authors propose a precise definition of a *benchmark* program. Our ML applications don't correspond to this definition, and we don't have this objective. However, their article offers pertinent recommendations on the type of optimizations to apply or avoid when measuring the performance of quantum codes: (1) apply optimizations with *constant time cost* (e.g. : generate final circuits with quantum gates adapted to the target hardware), (2) beware of optimizations that could exponentially increase the number of quantum gates and limit scalability (which could occur when generating error mitigation to increase output quality), and (3) never perform optimizations based on knowledge of the circuit's expected output.

We have adopted rules 1 and 2 by checking the configuration of the qiskit environment. As for rule 3, we are conducting quantum ML experiments and, of course, adjusting our system hyperparameters and learning algorithm to improve our circuit result. But we do not change the topology of our quantum circuit or the nature of our quantum gates to favor the exact result we need.

B. Performance modeling and metrics

The evaluation of the performance of quantum algorithms is a very hot topic as the search of an advantage over classical computers is ongoing. One part of this lies in the development of interesting metrics to evaluate quantum devices. There has been some work in this direction, notably the "CLOPS" metric developed by IBM [2] aiming at providing an overview of the speed of a QPU.

As real QPUs are being installed into more labs and computing centers, there is a growing interest in benchmarks for quantum computing. These provide insights into the real-world performance of QPUs and thus are of real scientific and industrial interest. In particular, comprehensive applicative benchmarks that give an overview of real workflows on QPUS are of interest [3].

A final major part of the evaluation of the performance of quantum computing lies in performance models. These allow projections and predictions for the performance of current and future QPUs and better interpretation of benchmarks and experiments. Some simple execution time models are starting to appear for the execution of classes of quantum circuits [4]. There is however little information regarding the evolution of execution time based on QPU occupancy. We'll assume in this work that the execution time of a quantum circuit is constant regardless of the number k of parallel executions of this circuit on a QPU.

C. Quantum multi-programming

Filling a QPU with independent circuits to be executed in parallel is widely exploited in multi-programming approaches and tools, such as *palloq* [5], *QuMC* [6] and *QuCloud+* [7]. Their approach consists firstly in gathering independent circuits to be executed (located in a task queue, for example) and grouping them into a single macro-circuit. Next, these tools look for the best mapping of qubits on the QPU architecture, so as to execute them in parallel while minimizing the noise generated and undesirable interference between qubits on NISQ processors (crosstalk).

Our research focuses on QML algorithms based on neural networks or generative networks, with the characteristic of executing the same circuit (one size, one depth) over and over again on different data, within a VQA-type iterative algorithm. So, to meet the needs of the same application and the same algorithm, we fill the QPU with identical circuits in order to make the best use of the QPU and, above all, to speed up learning calculations.

III. Parallel Quantum ML application

A. VQA and generative models for data clustering

To achieve data clustering on a CPU-QPU hybrid architecture, we use a Variational Quantum Algorithm (VQA) illustrated on Fig. 2. It is composed of:

1) a *quantum circuit Born machines* (QCBM) [8]: a quantum circuit implementing a *generative neural network* that takes advantage of the probabilistic nature of quantum physics and can run on a QPU (see Fig. 1),
2) an *Expectation-Maximisation* (EM) iterative algorithm [9], tailored to the training of probabilistic models for clustering applications.

See [10] for more details on this hybrid algorithm.

B. Design of quantum macro-circuits with qiskit

To merge elementary circuits in a macro circuit we follow the principles of Algorithm 1. First, we create an empty macro circuit of the desired size, and an empty list of parameter sets. Next, we add each n elementary parametric circuit to the macro circuit and its parameter set to the global parameter list.

However, implementing this algorithm in Qiskit requires a few technical adaptations, described in the listing 1. As there was no qiskit module capable of automatically parallelizing multiple machine learning circuits, we modified the `BackendSampler` class for this purpose. The quantum model

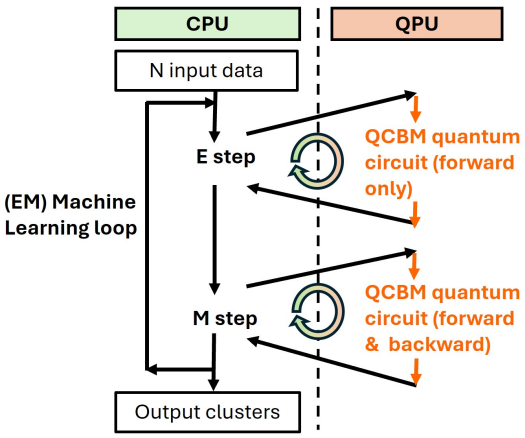

Fig. 2. VQA to control generative networks for data clustering

communicates circuits to the `BackendSampler`, which prepares them for the backend before sending them. This includes transpiling and mapping virtual qubits onto real qubits.

To parallelize our original elementary circuits, we create macro-circuits the size of our backend (listing 1, line 12) and place our elementary circuits in them as soon as they are transmitted to the `BackendSampler` (lines 16-30). We then let the `BackendSampler` take care of these circuits until the post-processing operation, during which we recover the result of the original circuits using a marginal counting function.

```
1  def fusion_circ(
2          self,
3          circuits: tuple[QuantumCircuit, ...],
4          parameter_values: tuple[tuple[float, ...], ...]):
5      # Number of qubits to uses
6      N = self.nb_qubits
7      # Number of qubits of original elementary circuits
8      n = circuits[0].num_qubits
9      # Number of parameters of original circuits
10     nb_param = len(parameter_values[0])
11     # New qiskit circuit creation (macro-circuit)
12     qc = QuantumCircuit(N,N)
13     param = [] # tuple for new parameters
14     i=0
15     # Add each elementary circuit into a macro-circuit
16     for circuit in circuits:
17         circuit_aux = circuit.copy()
18         # remove measurement
19         circuit_aux.remove_final_measurements()
20         # creat new set of qiskit parameters
21         new_p = ParameterVector("p"+str(i), nb_param)
22         # add the parameters to our circuit
23         circuit_aux = circuit_aux.assign_parameters(new_p)
24         # add original circuit to final macro-circuit
25         qc = qc.compose(circuit_aux, [i*n, (i+1)*n-1])
26         # add measure
27         qc.measure([i*n, (i+1)*n-1], [i*n, (i+1)*n-1])
28         # add parameters to our list
29         param += parameter_values[i]
30         i+=1
31     return qc, param
```

Listing 1. function to fusion qiskit parametric circuits in BackendSampler

IV. Test bed configuration

A. Quantum hardware

IBM offers the possibility of launching quantum code thanks to our jobs on real hardware: 10 minutes per month are available free of charge on a 127 qubits computer. In our case, among the different hardware available in May-June 2024 we choose the *ibm_brisbane*, Eagle r3 processor. We also experimented on two others IBM machines with the same architecture: *ibm_kyoto* and *ibm_sherbrook*.

Algorithm 1 Principle of elementary circuit fusion

Require: N ▷ size of macro-circuit
$circuits$ ▷ list of n identical elementary-circuits
$parametres$ ▷ list of elementary-circuits's parameters
Ensure: $macroCircuit$ ▷ macro-circuit
$macroParameters$ ▷ list of macro-circuits's parameters
$macroCircuit \leftarrow newQuantumCircuit(N)$
$macroParameters \leftarrow []$
for $i \leftarrow 1, \ldots, n$ **do**
$macroCircuit.compose(circuits[i])$
$macroParameters.insert(parametres[i])$
end for

B. Qiskit configuration

In order to send our circuits to the backend, qiskit has to go through several steps[1] during the transpilation process. In our case, all we need to do is configure the backend so that our circuits can be adapted by the transpiler with the correct default parameters (see listing 2).

```
1  #Adaptation to the backend (quantum hardware)
2  backend = service.get_backend('ibm_brisbane')
3  sampler = BackendSampler(backend)
4  #Quantum neural network definition
5  qnn = SamplerQNN(circuit=qc,sampler=sampler,...)
6  #Connection to pytorch framework
7  model_pytorch = TorchConnector(qnn,...)
```

Listing 2. from quantum backend to neural netowork to pytorch

The entire list of default transpiler parameters is available in qiskit online documentation[2]. By default, `resilience_level = 1`, which means *Minimal mitigation costs: Mitigate error associated with readout errors*[3].

V. Experiments and modeling

A. Experimental measurement of execution times

The solid lines in figure 3 show the experimental execution times of *forward* computations on the QPU of one of our generative models (there is one per cluster), using macro-circuits of 8 elementary circuits (16 qubits) up to 63 elementary circuits (126 qubits) and for three different numbers of shots. We measured only the time spent in forward computations of one E-step (for one model), running 2016 elementary circuits (to generate a distribution of 2016 data). We have not measured the execution time of an M step (backwards calculations), as this requires the same elementary circuit to be executed many times over. We would therefore have obtained the same curve profiles, and ultimately the same accelerations, but we

[1]https://docs.quantum.ibm.com/api/qiskit/transpiler
[2]https://github.com/Qiskit/qiskit/blob/stable/1.1/qiskit/compiler/transpiler.py
[3]https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.options.Options

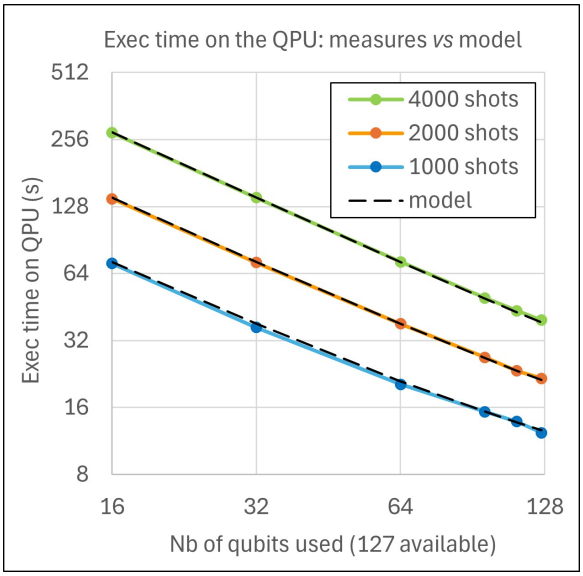

Fig. 3. QPU execution time of parallel macro-circuits performing E-step forward computations: measurement *vs* model

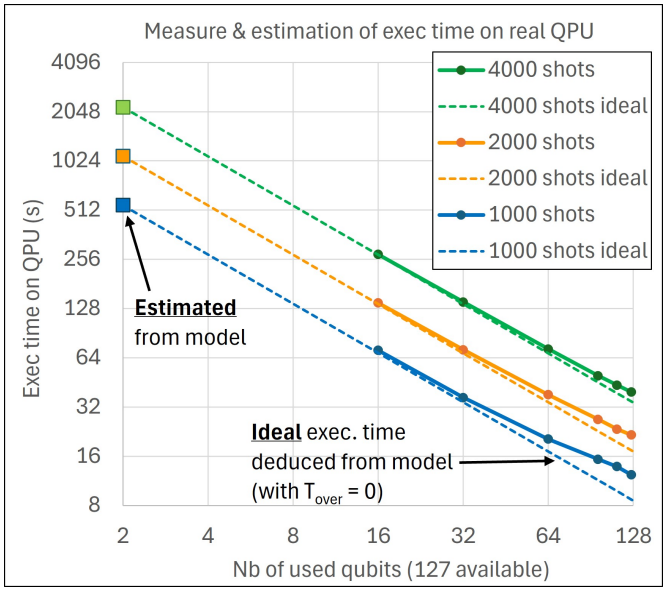

Fig. 4. Extended execution time of parallel macro-circuits performing E-step forward computations

wouldn't have been able to carry out these experiments with a QPU free time quota of 10 minutes per month.

We can observe significant and regular reductions in execution time (in logarithmic scales) when using macro-circuits, and dashed black lines are curves deduced from our execution model of macro-circuits, see section V-B. Parallel execution of several elementary circuits looks very interesting, even when the QPU is fully utilized and we need to compute the speedup reached, see section V-D.

Nevertheless, we need to investigate whether the data clustering is affected by this total use of the QPU, or at least whether the distribution of points generated by forward calculations in an E-step is not degraded, see section V-C.

B. Modeling performances with only elementary circuits

To compute the speedup we would need to measure execution time using only elementary circuits (i.e.: with macro-circuits including only one elementary circuit on 2 qubits). However, the total execution time would be too long. These experiments are out of our reach and would preclude using the QPU for other, more useful calculations. Instead, we decided to establish a model of execution times using only 2 qubit elementary circuits, compatible with our measurements on 16 to 126 qubits and with the operation of qiskit described in the IBM documentation.

We come up with a model to predict the execution time of the quantum jobs with different levels of space parallelism for the elementary circuits. As a reminder, during a job are executed a certain number N_{circ} of circuits, each sampled N_{shot} times. We take two hypothesis here: shot durations are constant and there is a fixed overhead to running a job[4].

[4]https://docs.quantum.ibm.com/run/execution-modes#best-practices

This leads to the simple execution time model for 1 job (with numerical values compatible with IBM documentation):

$$T_{job} = T_{over\ (4s)} + N_{circ} \cdot N_{shot} \cdot T_{shot\ (270\mu s)} \quad (1)$$

Moreover, the number of executed circuits N_{circ} depends on the chosen parallelism –ie how many elementary circuits we pack on the QPU– as well as the number of qubits per elementary circuit. We define N_{circ}^{elem} the number of elementary circuits to run requiring N_{qubits}^{elem} qubits each, and N_{circ}^{macro} the number of macro-circuits to run requiring N_{qubits}^{macro} qubits each. The total number of used qubits remains unchanged:

$$N_{circ}^{macro} \cdot N_{qubits}^{macro} = N_{circ}^{elem} \cdot N_{qubits}^{elem} \quad (2)$$

By replacing N_{circ} with N_{circ}^{macro} we finally have:

$$T_{job} = T_{over} + \frac{(N_{circ}^{elem} \cdot N_{shot} \cdot T_{shot})}{(N_{qubit}^{macro}/N_{qubit}^{elem})} \quad (3)$$

This looks very similar to the execution time on a parallel processor following Amdahl's law [11] which has the following form:

$$T = T_{serial} + T_{parallelizable}/N_{thread} \quad (4)$$

The experimental curves in Fig 3 are superimposed with dotted lines calculated from our model. Experimental and model-based curves fit and tends to validate our model. So we used this model to deduce the execution time with macro-circuits including only one elementary circuit (2 qubits) that appear well in line with the experimental curves on Fig 4. Again these curves tend to validate our model based on IBM documentation, and allow us to compute the speedup reached by macro-circuits (see next section).

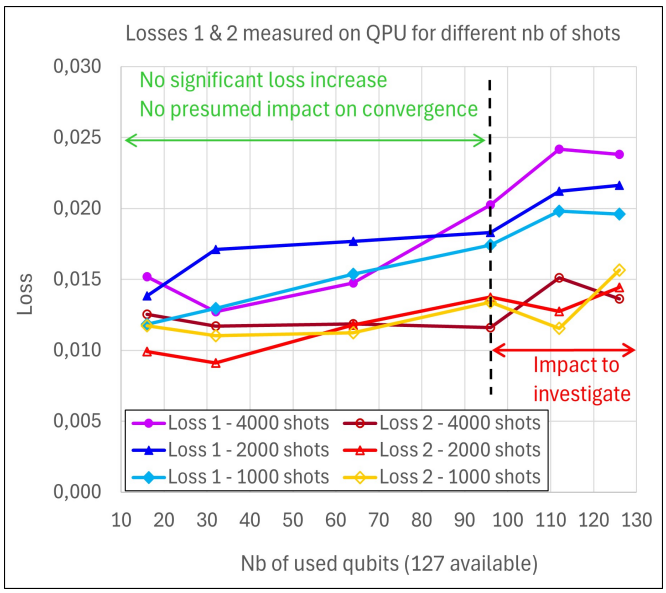

Fig. 5. Losses measured on a real QPU for different number of shots during E-step forward computations

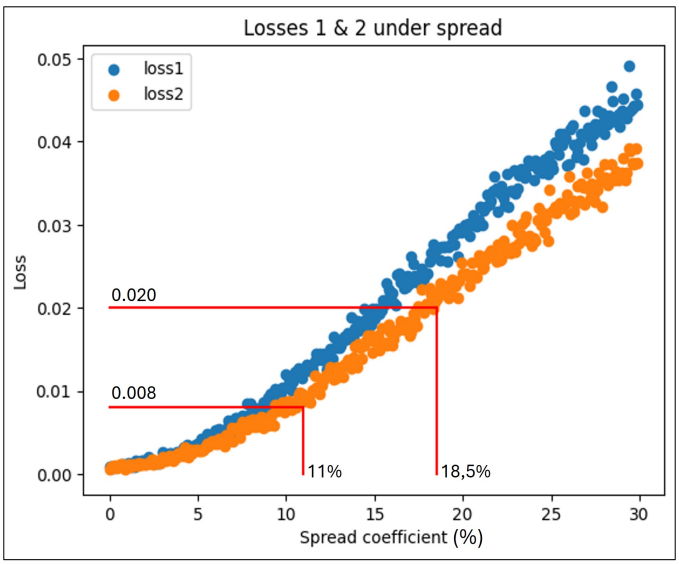

Fig. 6. Losses under Spread: when the loss is less than 0.02, the spreading factor is less than 18.5%

C. Experimental sensitivity of the loss

To measure the impact of noise, we calculate the *Maximum Mean Discrepancy* (MMD) loss [12] between the distribution generated by a perfect simulation and that of our overloaded QPU. This loss tells us the deformation between these two distributions, and thus helps us understand the impact of noise. Of our two generative models, one generates a trivial distribution (analyzed by loss 2) and the other a more complex one (analysed by loss 1), which explains the significant difference of the experimental loss curves on Fig. 5.

Experimental loss values of Fig. 5, deduced from results of forward computations of E-steps on a real QPU, show that:

- losses of the two generative models fluctuate not only with the number of qubits used, but also with the number of shots,
- up to 96 qubits (48 elementary circuits executed in parallel), the range of loss variation is limited and does not increase too much,
- but this range of loss variation increases further between 96 and 126 qubits.

Overall, Fig. 5 shows us that overloading the QPU has an impact on losses, which increase moderately but steadily. Consequently, overloading the QPU to parallelize elementary circuits may have an impact on the overall training of our model. The learning mechanism, and therefore the data clustering, could thus take longer to achieve the same result.

Further investigations remain difficult in the absence of longer access to QPUs. Nevertheless, we are studying in simulation the impact of greater loss on the distributions of our generative models, which then tend to disperse. To do this, we take the points of the perfect distribution, normalize them and center them at 0. We then multiply them by a spreading coefficient, and calculate the loss induced by this spreading. We have then linked the spread of our distributions with the loss, as illustrated in Fig. 6, and conducted a set of experiments mixing simulations and real experiments function of the execution time required:

- We managed to train our models in 2 qubit noisy simulations (with only one elementary circuit per macro-circuit). We measured a loss of 0.008 corresponding to a range spread of 11% (see Fig. 6)
- We conducted also short experiments on a real QPU using many qubits. On 96 qubits we measured a loss of 0.020 corresponding to a range spread of 18.5% (see Fig. 6).

However, it remains difficult to precisely link noise, losses and distribution distortions, and to set thresholds that must be respected to prevent too many losses and distortions from slowing down or blocking the learning process.

Unexpectedly, a study on noise injection [13] encourages us to pursue a more comprehensive use of QPUs. Noise injection is a technique used to improve the learning of generative models. However, the use of this technique with quantum generative models has not yielded any results [14]! We can therefore assume that small variations in noise have no significant impact on generative models.

So, for the time being, based on our current experience and observations, we assume that a dispersion of less than 20% does not significantly impact the learning of our generative models. But more comprehensive and time-consuming experiments on different QPUs are required.

D. Speedup reached using macro-circuits

Considering the execution time with a single elementary circuit per macro-circuit, deduced from our time model presented in section V-B and illustrated in figure 4, and the different execution times measured on the IBM 'brisbane' quantum machine, we can estimate a speedup as a function of the

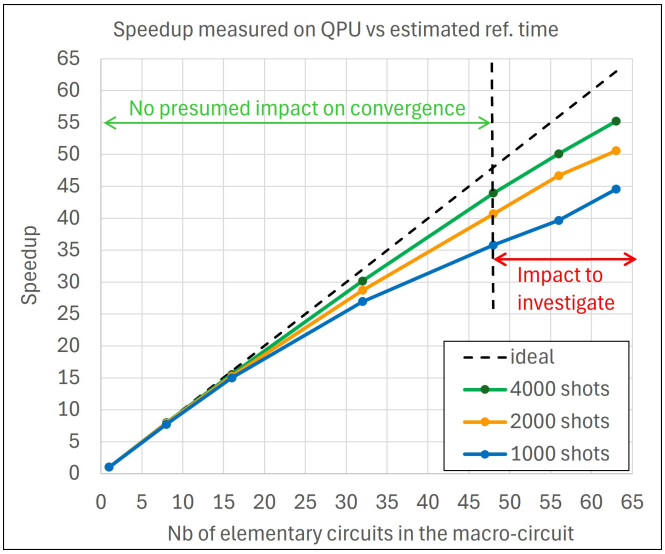

Fig. 7. Speedup achieved on a QPU with macro-circuits *vs* a single elementary circuit, during forward computation of an E-step

number of elementary circuits included in each macro-circuit. Figure 7 shows the acceleration curve obtained during forward computation of an E-step on a real QPU.

As we explained in section V-A, we limited our experiments and time measurement to the forward computation of E-steps to respect our time quota on QPU (10 minutes per month). However, the longer M-steps launch the same quantum circuits on the QPU and would reach the same speedup. Consequently, figure 7 shows the accelerations achieved on the IBM 'brisbane' QPU for the whole quantum part of our data clustering hybrid application.

As explained in section V-C, based on our current analysis of the impact of noise on our generative models, we consider these accelerations to be reliable up to 48 elementary circuits (96 qubits) executed in parallel in each macro-circuit. We should therefore be able to achieve accelerations of up to $[35-45]$ by parallelizing the execution of elementary circuits within a QPU, as in a multi-core processor in conventional computing. Depending on the noise generated, this acceleration could then reach $[45-55]$ on this Machine Learning application.

VI. Recommendation and perspective

As QPUs are currently scarce resources, following the significant speed-ups obtained in this study ($[35-45]$), we recommend exploiting replication parallelism within the QPU whenever possible in Machine Learning applications without disrupting their computations. This would enable a greater number of learnings to be performed on the same QPU. The code for the Qiskit extension is easy to develop (see Listing 1).

However, from a technical point of view, we've only experimented with 2-qubit quantum circuits, and it's easy to find 2 qubits close together and capable of supporting entanglement in a QPU. It therefore seems quite straightforward to install a multitude of totally independent 2-qubit elementary circuits in a QPU. We now plan to reproduce our experiments with 4-qubit QCBMs (to work on 4-dimensional data) and evaluate the feasibility, speed-up and quality of results of calculations up to 126 qubits.

From a machine learning point of view, a recent study [14] showed that other generative networks, the *Quantum Generative Adversarial Networks* (qGAN), could be trained successfully in the presence of noise. They could therefore support more intensive parallelism within the QPU. However, the authors of this study also highlighted the sensitivity of hyperparameters and the need to mitigate them when training on NISQ. We are currently studying qGAN parallelization and comparing it with QCBM.

References

[1] M. Amico, H. Zhang, P. Jurcevic, L. S. Bishop, P. Nation, A. Wack, and D. C. McKay, "Defining best practices for quantum benchmarks," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Bellevue, WA, USA, sep 2023, pp. 692–702.

[2] A. Wack, H. Paik, A. Javadi-Abhari, P. Jurcevic, I. Faro, J. M. Gambetta, and B. R. Johnson, "Quality, Speed, and Scale: Three key attributes to measure the performance of near-term quantum computers," Oct. 2021.

[3] T. Lubinski, S. Johri, P. Varosy, J. Coleman, L. Zhao, J. Necaise, C. H. Baldwin, K. Mayer, and T. Proctor, "Application-Oriented Performance Benchmarks for Quantum Computing," *IEEE Transactions on Quantum Engineering*, vol. 4, pp. 1–32, 2023.

[4] J. Weidenfeller, L. C. Valor, J. Gacon, C. Tornow, L. Bello, S. Woerner, and D. J. Egger, "Scaling of the quantum approximate optimization algorithm on superconducting qubit based hardware," *Quantum*, vol. 6, 2022.

[5] Y. Ohkura, T. Satoh, and R. Van Meter, "Simultaneous execution of quantum circuits on current and near-future NISQ systems," *IEEE Transactions on Quantum Engineering*, vol. 3, 2022.

[6] S. Niu and A. Todri-Sanial, "Enabling Multi-programming Mechanism for Quantum Computing in the NISQ Era," *Quantum*, vol. 7, 2023.

[7] L. Liu and X. Dou, "QuCloud+: A holistic qubit mapping scheme for single/multi-programming on 2D/3D NISQ quantum computers," *ACM Transactions on Architecture and Code Optimization*, vol. 21, 2024.

[8] C. Riofrío, O. Mitevski, C. Jones, F. Krellner, A. Vučković, J. Doetsch, J. Klepsch, T. Ehmer, and A. Luckow, "A performance characterization of quantum generative models," 01 2023, ArXiv.

[9] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the royal statistical society: series B (methodological)*, vol. 39, no. 1, 1977.

[10] J. Rauch, D. Rontani, and S. Vialle, "Generative-based algorithm for data clustering on hybrid classical-quantum nisq architecture," in *37th GI/ITG International Conference on Architecture of Computing Systems*, Postdam, Germany, May 2024, to appear.

[11] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, 1967, pp. 483–485.

[12] S. Paik, M. Celentano, A. Green, and R. J. Tibshirani, "Maximum mean discrepancy meets neural networks: The radon-kolmogorov-smirnov test," 2023. [Online]. Available: https://arxiv.org/abs/2309.02422

[13] R. Feng, D. Zhao, and Z.-J. Zha, "Understanding noise injection in gans," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 3284–3293. [Online]. Available: https://proceedings.mlr.press/v139/feng21g.html

[14] K. Borras, S. Y. Chang, L. Funcke, M. Grossi, T. Hartung, K. Jansen, D. Kruecker, S. Kühn, F. Rehm, C. Tüysüz, and S. Vallecorsa, "Impact of quantum noise on the training of quantum generative adversarial networks," *Journal of Physics: Conference Series*, vol. 2438, no. 1, 2023.