Quantum Benchmarking

Robert Schuh
Schwerpunktmodul activity in the Quantum Computing course

Benchmarking

Introduction

A new algorithm to benchmark quantum processors was created and tested on a variety of systems. After the initial approach proved to have some issues, a modified version of the algorithm was tested as well.

IBMQ and Quantum Brilliance

Algorithms are tested on emulated versions of IBM Quantum [1] and Quantum Brilliance [2] systems. The error data of several IBMQ processors available on their free tier is requested through their API to simulate quantum circuits in Qiskit on backends that approximate the IBMQ systems.

Using the Quantum Brilliance Emulator software, the same quantum circuits are simulated on several Noise Models available from the Quantum Brilliance servers. The benchmarking algorithms determine how the circuits are generated and how the measured counts are processed. Running the algorithms on real quantum processors would be possible, but was not attempted due to long wait times when using a free IBMQ account.

First approach

The first algorithm tested worked as follows:

1. Generate n_c random circuits with n_q qubits and n_g gates.
2. Run the random circuits both on a perfect error-free simulator and the one that is being benchmarked for n_s shots.
3. Compute the measurement histograms as vectors, normalized to the number of shots with $\sum_i p_i = 1$.
4. Calculate an error as euclidean distance between the perfect and benchmarked measurements, $\sqrt{\sum_i (p_i^{\text{perfect}} - p_i^{\text{benchmark}})^2}$.

This error score is 0 if the benchmarked simulator matches the error-free one, and up to $\sqrt{2}$ if it is confidently incorrect, for example if the error-free simulator consistently returns the "00" state but the benchmarked one consistently returns "01" on a circuit.
As n_c random circuits are tested, the result is an array of error score, with the mean and standard deviation plotted.

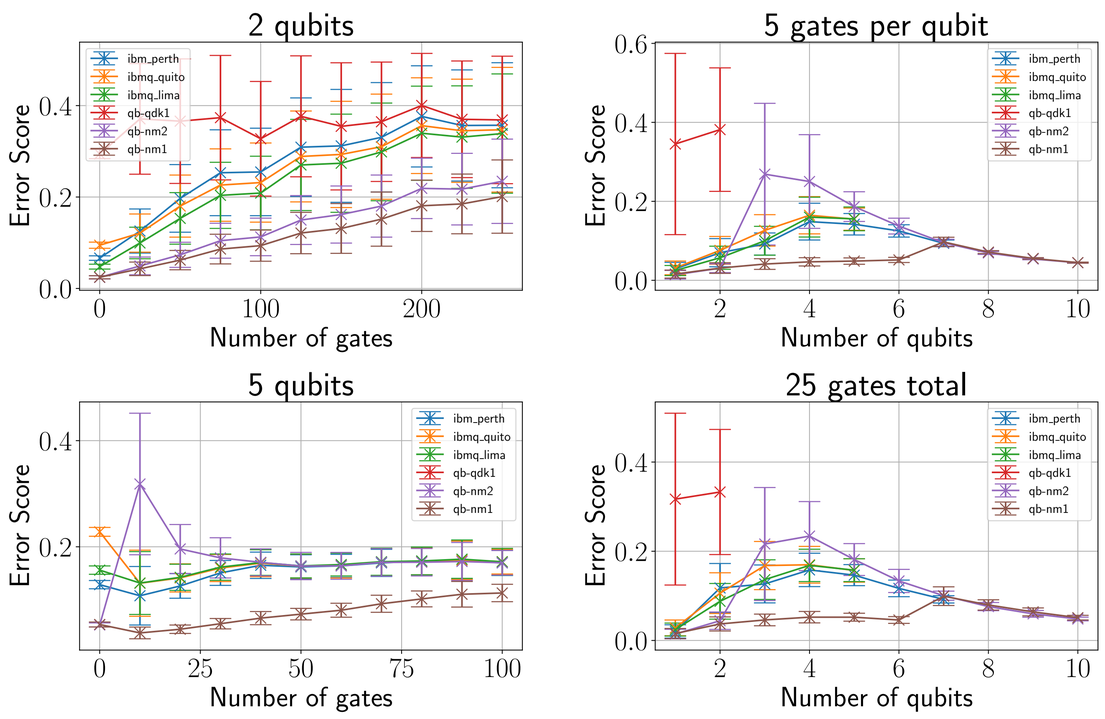
Second approach

After the failure of the first approach, the second approach is a modification of the first algorithm.

1. Generate n_c random circuits with n_q qubits and floor$(n_g/2)$ gates.
2. For each random circuit, append its inverse.
3. Run the random circuits on a perfect error-free simulator to verify that it measures the initial state.
4. Run the random circuits on each emulator that is being benchmarked for n_s shots.
5. Compute the measurement histograms, normalized to the number of shots with $\sum_i p_i = 1$.
6. Calculate he error score as the sum of all p_i that are not the initial state.

The resulting error score is the probability of measuring a result other than the correct one. For random results, it approaches $1 - 0.5^{n_q}$.

Results of first approach

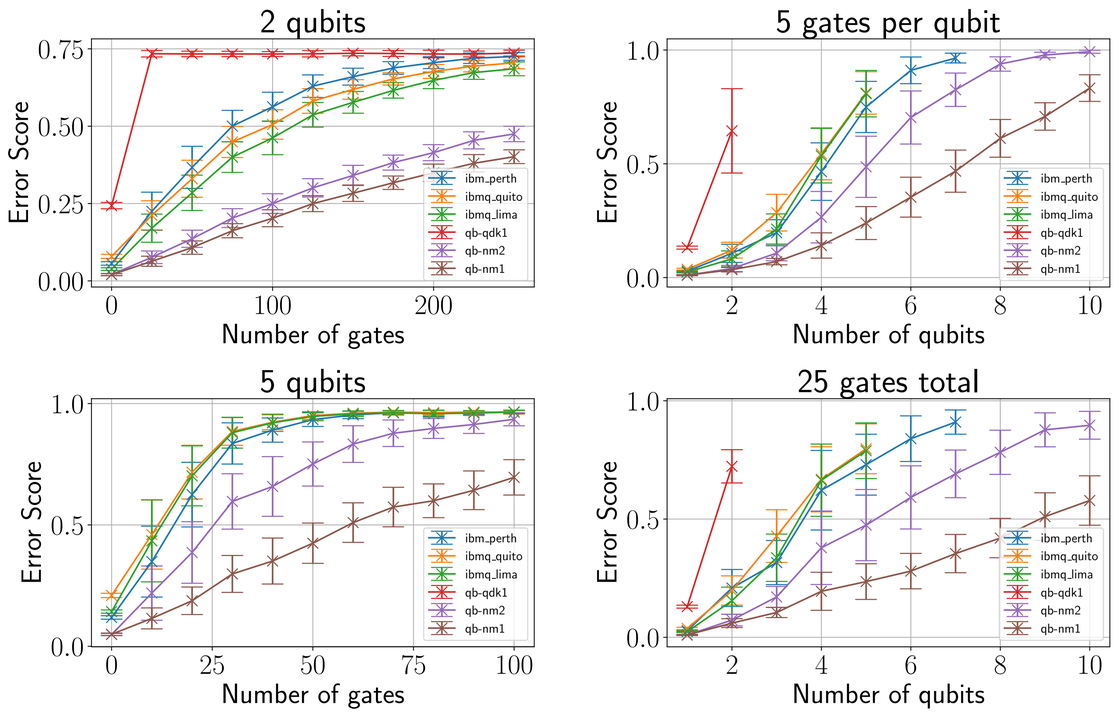


The first algorithm is tested on all the IBMQ and QB emulators for $n_c = 50$ random circuits each, $n_s = 2000$ shots and different numbers of qubits and gates.

Problems

As immediately apparent in the figure above, the first algorithm does not work as desired. For larger numbers of qubits, all the error scores of all the emulators converge and go down. This is because for random circuits, the histograms tend to approach even distributions, and any errors in the quantum computers simply cause more randomness, making the result even smoother.
Therefore, the result of the first algorithm is not usable to determine how good a given quantum computer is at running circuits.
Some improvement is required.

Results of second approach



The second approach is tested on the same backend as the first one, with the same numbers of shots, random circuits, qubits and gates.
The results are usable: We can see that "qdk1" can only handle 2 qubits and struggles with even very few gates, "nm1" and "nm2" can handle lots of qubits and gates, and all the IBM systems are fairly close, with "quito" and "lima" being able to handle 5 qubits while "perth" can do 7.
The error scores also always go up as gate count increases, as would be desired in a method to benchmark quantum chips.

Notes on implementation

As both algorithms only send a circuit and a desired number of shots to either the emulated IBMQ or the QB system and only take a histogram of counts back, they could easily be used on real machines. However, this was not done due to lack of access to real machines without significant wait times, as the queue on IBM systems for non-paid access is several hours long.

References

[1] IBMQ Resources, *https://quantum-computing.ibm.com/services/resources*, 2023
[2] QB Documentation, *https://qristal.readthedocs.io/en/latest/rst/noise_models.html*, 2023