Quantum Circuit Simulation Benchmark Guide on Cirq,Qiskit,QSharp, and

Samuel A. Stein, Pacific Northwest National Laboratory
samuel.stein@pnnl.gov

May 28, 2021

Abstract

The following document is an outline on how to benchmark a Quantum circuit's run time using Python. Four quantum programming languages simulators are bench marked, namely Cirq,Qiskit,QSharp and Quil. The document is outlined such that pre-requisite installs are described.

Contents

1 Pre-requisite Installation

1.1 System Pre-requisite Installation

The benchmarking system outlined runs on the following technologies:

- Python 3.7 and above
- Bash
- Anaconda
- Quil servers
- QSharp compiler
- Python Quantum Processing packages
- Javascript package q-convert

Before any of the quantum programming languages are prepared, ensure that python 3.7 is installed on your system, anaconda is installed and both of these technologies are working within a BASH terminal.

1.2 Installing Quil

Quil requires a Rigetti Quantum Virtual Machine (QVM) and a Rigetti Quil Compiler (quilc) to perform simulations. The Virtual Machine allows for the High Performance simulation, and the compiler allows for the compilation and optimization to native gate sets.

1.2.1 Installing qvm and quilc on Windows and macOS

Download the applicable Forest SDK from the link: `https://qcs.rigetti.com/sdk-downloads`. Once this is downloaded, run and install the application.

1.2.2 Installing qvm and quilc on Linux

Using the link `https://qcs.rigetti.com/sdk-downloads`, download the applicable distribution for your linux (be it deb or rpm). In the below code, the

```
tar -xf DOWNLOADED_FILE_NAME
cd DOWNLOADED_FILE_NAME
sudo ./DOWNLOADED_FILE_NAME.run
# We verify the instillation here
qvm --version
quilc --version
```

1.2.3 Launching Quil Servers

Quil is reliant on the QVM and QUILC servers. To do so, launch two consoles (you can use tmux here) and run the following code
In Console 1:

```
qvm -S
```

In Console 2 run:

```
quilc -S
```

Do not close these servers. They must be running the whole time evaluations are being performed.

1.3 Q Sharp Installation

To set up QSharp, we make use of their conda enviroment. To do this, run the following lines of code in a new terminal (separate from the 2 servers running)

```
conda create -n qsharp-env -c quantum-engineering qsharp notebook
conda activate qsharp-env
```

Once this has run, your terminal will be operating in the qsharp virtual enviroment. Move your directory to your working directory where your QASM circuits are.

1.4 Installing q-convert

To install q-convert, you must have npm installed prior. To install npm, install node.js from the following link `https://www.npmjs.com/get-npm`. Once this is installed, verify your installation by typing

npm

If this works succesfully, run the following command:

```
npm -g install q-convert
```

1.5 Python Dependancy Installing

To install Qiskit and Cirq, and other dependencies we can make use of pip install.

```
pip install qiskit
pip install cirq
pip install matplotlib
pip install pyquil
pip install qsharp
```

These packages will install all the required python packages, package converters, and extensions on top of the qsharp and quil backends.

2 How to benchmark a .qasm file

2.1 File Structure

The system is divided into the files main.py, qs-bench.py, append-evaluation.py, qiskit-bench.py. The roles of each file are as follows:

- Main.py - The main python file that is called to evaluate the .qasm file. Within this file, the q-evaluation package is used to convert the .qasm file into a compatible Cirq python file, a Quil python file, and a QSharp .qs file. A Qiskit file does not need to be created, as Qiskit has diverse support for of loading .qasm files, unlike the other technologies.
- qs-bench.py - This python file is called from main.py, and reads in the .qs file that was created. It is then simulated and evaluated, where a results run speed is written to a .txt file in a results/circuit name/ sub directory.
- append-evaluation.py - The python files generated from q-convert do not contain the code required for timing the simulation. Hence, a python file edits these generated cirq and quil python files to inject a section of code that allows for the timing of simulations.
- qiskit-bench.py - This Python file is run from main.py, but is simply a Qiskit file that reads in the qasm and benchmarks it. Making use of transpile and assemble, Qiskit is accepting of most QASM formats.

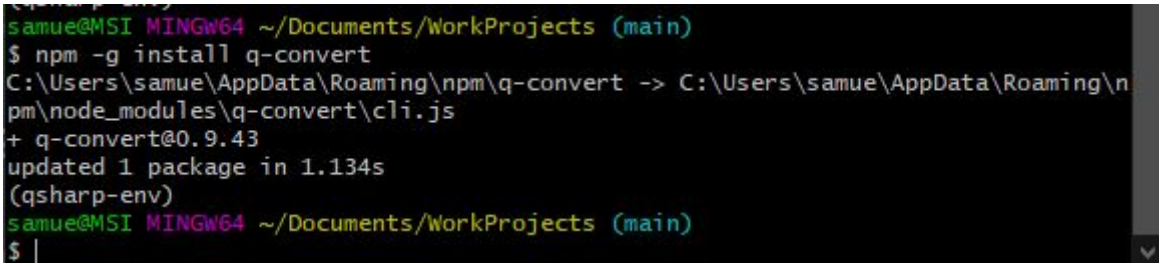

Figure 1: Installing q-convert

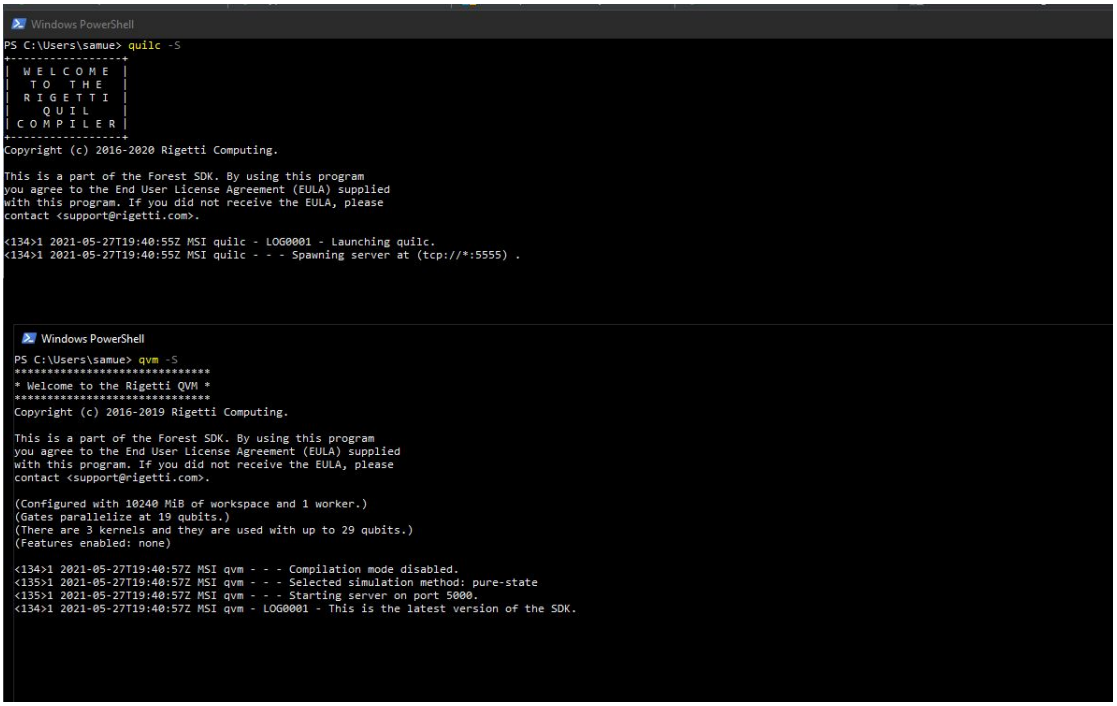

Figure 2: Launching QSharp Backend

2.2 Running the script

To benchmark a .QASM file, place all of the .py files within a directory, and optionally place the .qasm file in the directory as well. Following this, whilst in the conda bash, run the following command when evaluating NAME.qasm.

```
python main.py NAME.qasm
```

Following this, a directory "resutls" will be generated. Within this folder, a folder with NAME is created and all of the results are stored within this directory.

3 Example Benchmark

3.1 Benchmarking a 4-Qubit Cat State Circuit

As a concrete example, the following section outlines the steps above with the results shown.

1. Install the q-convert javascript package
2. Launch Quil servers using the code: "qvm - S" in Console 1 and "quilc -S" in Console 2.
3. Launch Conda Enviroment

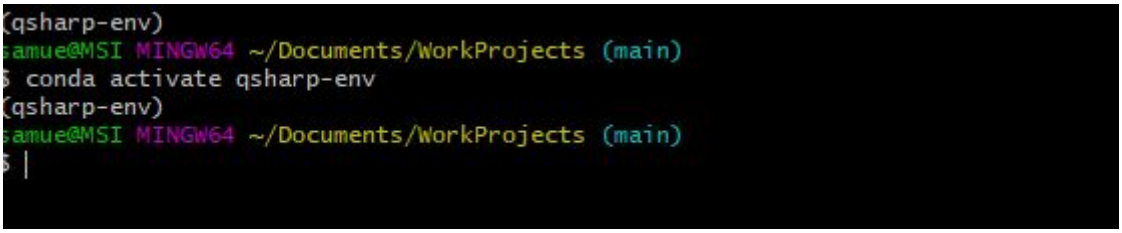

Figure 3: Conda Enviroment Backend

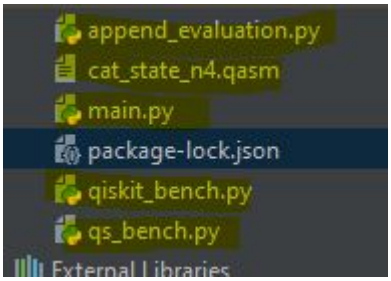

Figure 4: Sample Directory

4. Set up your directory with all the python files in one directory, and place your .qasm file in the directory

5. Run the following command: python main.py cat_state_n4.qasm. Following this, a "result" folder with "cat_state_n4" as a sub-directory containing all of the results is created.

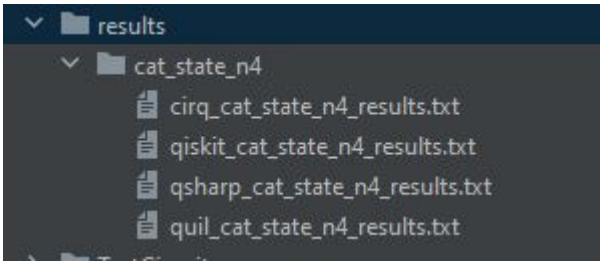

Figure 5: Results