Parameterising the circuit-

The simulator does not take in the number of qubits as an input. The simulator only takes in the initial state of the qubit and the QASM file that is to be simulated and returns the states after implementation of each of the gates in the QASM file and the final state after the simulation. The number of qubits is a parameter that is defined in the grammar of the simulator, that is, the universal set of gates that are defined for the simulator. The number of qubits of operation of the gates has been defined to 16 since all of the qubit registers of the QASM files read qreg q[16].

The way to change the number of qubits of operation of the simulator is by changing the number of qubits in the functions by changing the value of variable ‘q’ (which has been initialized to 16 in for the submitted simulator) in the functions of the gates that have been defined in the simulator.

Testing the simulator-

The 14 benchmark QASM files given to us were used for testing the simulator. The benchmark files were run on an already existing simulator, the Qiskit simulator and the outputs were noted for the individual files. The qiskit “aer_simulator” was used with the input state being all ‘0’s . The simulator was then run for the benchmark input files and the results were matched with the output of the qiskit simulator.

The individual gates were first tested before running the whole simulator to make sure that the gates were working correctly to make testing smooth. Different 1 and 2 qubit input states were used to test the gates.

****Output of the simulators are in reverse order to what the outputs were found for the qiskit simulations as the order of qubits is reversed in my simulator.**

QASM File	Output
`Miller_11.qasm`	`(1+0j)|0000000000000000>`
`decod24-v2_43.qasm`	`(1+0j)|0001000000000000>`
`one-two-three-v3_101.qasm`	`(1+0j)|1100000000000000>`
`hwb5_53.qasm`	`(1+0j)|0000000000000000>`
`alu-bdd_288.qasm`	`(1+0j)|0000001000000000>`
`f2_232.qasm`	`(1-0j)|0000011000000000>`
`con1_216.qasm`	`(1-0j)|1001011100000000>`
`mini_alu_305.qasm`	`(1+0j)|0000000110000000>`
`wim_266.qasm`	`(1-0j)|1101111111100000>`
`cm152a_212.qasm`	`(1-0j)|0111000000000000>`
`squar5_261.qasm`	`1-0j)|0000000011011000>`
`sym6_316.qasm`	`(1+0j)|0000000000001000>`
`rd84_142.qasm`	`(1+0j)|0000000000000000>`
`cnt3-5_179.qasm`	`(1+0j)|0000000000000000>`

Table1. Different QASM benchmark files and their outputs.

Scalability with number of qubits-

The simulator was run twice. Once such that the states of the qubits were printed after application of each of the gates. The python environment approximates the value of $\frac{1}{\sqrt{2}}$ and does not use the exact value. Hence, the final states without any post-processing gives a whole bunch of other states with negligible amplitudes in the form of error. To get rid of the states the amplitudes of each of the states were rounded off to 4 digits. The rounding of and the printing the states after execution of the gates significantly increases the execution time. Below are given two tables and respective plots for execution with and without printing the states after application of each gates.

QASM File	Number of Qubits used	Number of lines of code	Execution Time (in seconds)
`Miller_11.qasm`	3 qubits	54 lines	4.93
`decod24-v2_43.qasm`	4 qubits	56 lines	5.16
`one-two-three-v3_101.qasm`	5 qubits	74 lines	6.91
`hwb5_53.qasm`	6 qubits	1,340 lines	131
`alu-bdd_288.qasm`	7 qubits	88 lines	8.23
`f2_232.qasm`	8 qubits	1,210 lines	118.1
`con1_216.qasm`	9 qubits	958 lines	94.1
`mini_alu_305.qasm`	10 qubits	177 lines	16.9
`wim_266.qasm`	11 qubits	990 lines	97.5
`cm152a_212.qasm`	12 qubits	1,225 lines	119.1
`squar5_261.qasm`	13 qubits	1,997 lines	199
`sym6_316.qasm`	14 qubits	274 lines	26.4
`rd84_142.qasm`	15 qubits	347 lines	32.7
`cnt3-5_179.qasm`	16 qubits	179 lines	16.6

Table2. Execution time **with** states being printed after each step

QASM File	Number of Qubits used	Number of lines of code	Execution Time (in milliseconds)
Miller_11.qasm	3 qubits	54 lines	226
decod24-v2_43.qasm	4 qubits	56 lines	229
one-two-three-v3_101.qasm	5 qubits	74 lines	267
hwb5_53.qasm	6 qubits	1,340 lines	2770
alu-bdd_288.qasm	7 qubits	88 lines	265
f2_232.qasm	8 qubits	1,210 lines	2470
con1_216.qasm	9 qubits	958 lines	2100
mini_alu_305.qasm	10 qubits	177 lines	438
wim_266.qasm	11 qubits	990 lines	2220
cm152a_212.qasm	12 qubits	1,225 lines	2500
squar5_261.qasm	13 qubits	1,997 lines	4680
sym6_316.qasm	14 qubits	274 lines	623
rd84_142.qasm	15 qubits	347 lines	762
cnt3-5_179.qasm	16 qubits	179 lines	44

Table3. Execution time **without** states being printed after each step

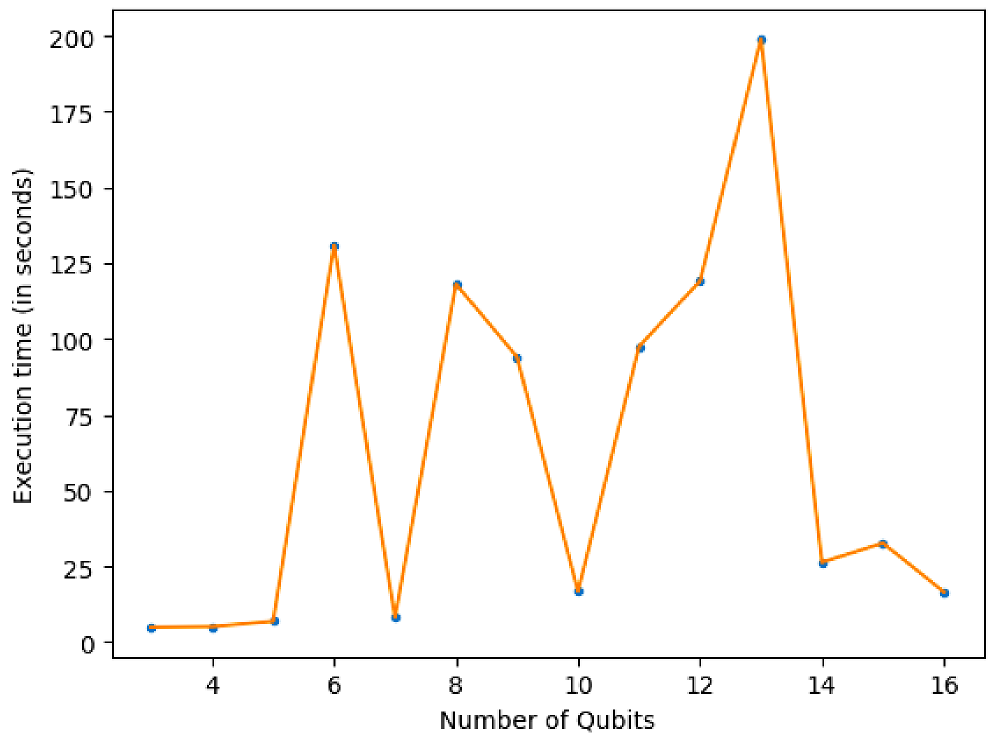

Figure 1: Execution time graph with number of qubits for states being printed after each step.

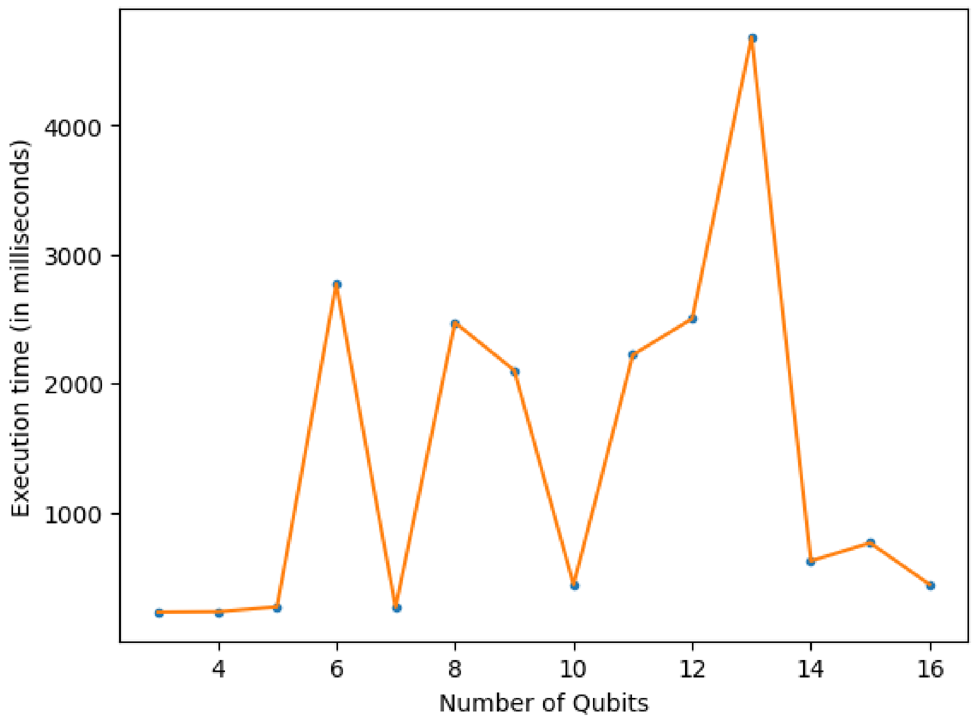

Figure 2: Execution time graph with number of qubits for states not being printed after each step.

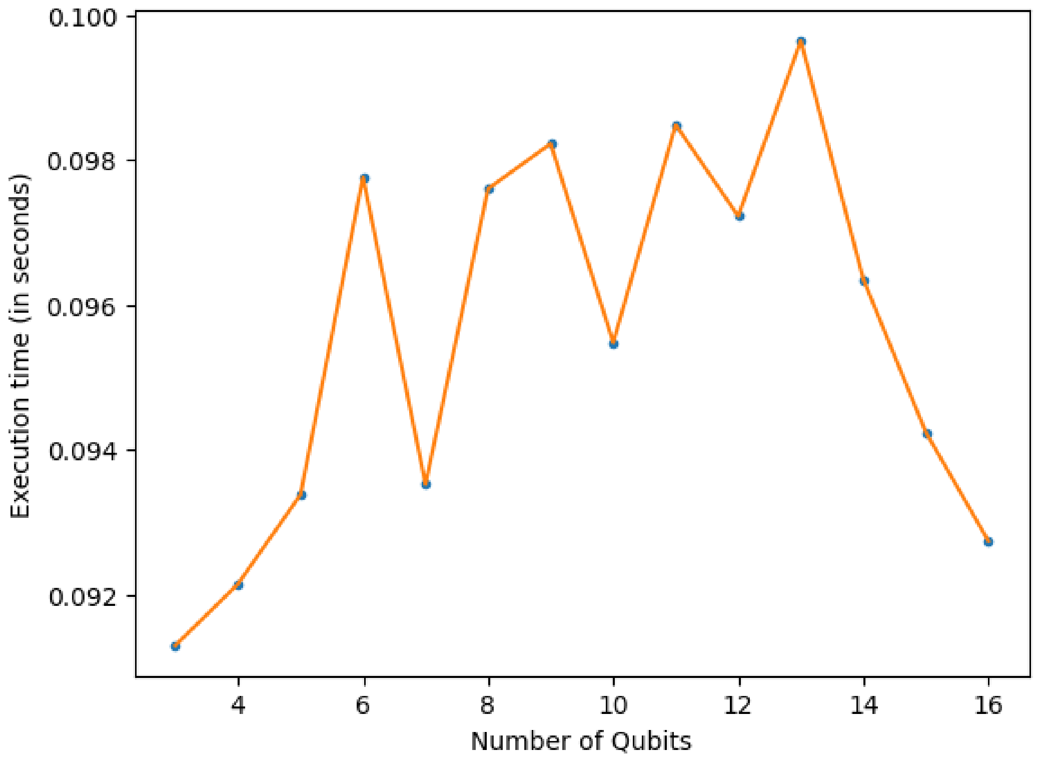

Figure 3. Execution time normalized for the number of lines of codes in each program.

Three graphs are plotted, one for the case when the states are being printed after each step of the simulation, the next for the case when only the final state is being printed and the last plot is a normalized plot where the execution time is divided by the number of lines of codes to check the scalability with the number of qubits.

As the number of qubits grows the execution time of the simulation also increases. But the execution time for the simulations performed is also very sensitive to the number of lines of code as the code is basically applying the quantum gates and increase in the number of lines of code increases the number of quantum gate implementations.