CS239 - UCLA

SPRING 2019

HOMEWORK 13 : REPORT

Using PyQuil and Rigetti's Quantum Computer for Quantum Algorithms

GROUP MEMBERS

Rishab Doshi - 305220397

Srishti Majumdar - 105225254

Vidhu Malik - 305225272

Quantum Algorithms

Part 1. Evaluation

2.0 Brief Summary of Solution

This section presents a brief summary of the implementation of the solutions for the four problems.

For Deutsch-Jozsa, Bernstein-Vazirani and Simon's algorithm, the utility functions used are mentioned in Table 1.0.1. The parameterization and encapsulation of the code is the same for all three algorithms.

Description	Function
Get decimal number from binary bit String	getDecimalNo(bitString)
Get qubit vector from the binary bit vector String	getQubitVector(bitString)
Get all possible bitstrings(2^n) for n bits	getAllPossibleNBitStrings(n)

Table 1.0.1: Common Util Functions

For solving Grover's, we make use of a function similar to getAllPossibleNBitStrings(n) called get_possibilities(n).

The quantum circuit, implementation strategy, generation of the Uf matrix, verification and post-processing steps remain the same as before (Homework 8 and 9) since we are reusing our PyQuil code for the assignment.

The following were the major changes made to our original PyQuil so that it works on the QMI instance:

1. Instantiating compiler using lattice instead of qvm.
2. Recognizing device qubits and using those qubits for the rest of the program.
3. Manipulating circuit to work based on required qubits.
4. Displaying results from only required qubits instead of all qubits from the lattice in consideration.

2.1 Results and Statistics:

In this section, we introduce our effort towards implementing the four algorithms on the quantum computer. Some general information and statistics like the minimum time taken by the program are provided. Exact statistics (like the variation in execution time based on function) are mentioned in the next few subsections.

Quantum lattices worked on:

1. 2 Qubits
2. 3 Qubits
3. 6 Qubits
4. 7 Qubits
5. 8 Qubits
6. 12 Qubits

Quantum Algorithms and Variations Tried:

1. Deutsch-Jozsa: Random Constant and Balanced functions, n= 1,2,...,11 (1 helper bit)
2. Bernstein-Vazirani: Functions with random values for a and b, n= 1,2,...,11(1 helper bit)
3. Simon's: Three variations for functions tried- f1 (s=110), f2 (s=001) and f3 (s=010), n= 1,2,...,6 (For each n, n helper bits)
4. Grover's: Two variations for function tried- random function with only one possible x such that f(x)=1 and random function with one or more possible x such that f(x)=1, n= 1,2,...,12 (No helper bits)

Minimum and Maximum Time Taken by each algorithm:

1. Deutsch-Jozsa: minimum- 0.47s, maximum- 9.00s
2. Bernstein-Vazirani: minimum- 0.45s, maximum- 0.74s
3. Simon's: minimum- 2.10s , maximum- 10.9s
4. Grover's: minimum- 0.15s , maximum- 5.90s [before timeout]

Note: For Deutsch- Jozsa, Bernstein-Vazirani and Simon's algorithms, the helper bit outputs are ignored.

Example Runs and Outputs:

```
********************
All qubits in this device in order: [0, 1, 2, 7, 13, 14, 15]
Key results  {0: array([0]), 1: array([0]), 2: array([0]), 7: array([0]), 13: array([0]), 14: array([0]), 15: array([0])}
CONSTANT
CONSTANT
N : 1 , FnType: CONSTANT, Time : 0.4192051887512207
********************
```

Figure 2.1.dj.1: Example Runs Shown for Deutsch-Jozsa

```
********************
All qubits in this device in order: [0, 1, 2, 7, 13, 14, 15]
N : 2 , A: 11, B: 1, Time : 0.7912521362304688
********************
```

Figure 2.1.bv.1: Example Runs Shown for Bernstein-Vazirani

```
********************
All qubits in this device in order: [1, 2, 10, 11, 14, 15, 16, 17]
N : 2 , QuantumCircuitCalls : 1, S: 01, Time : 2.463733434677124
********************
```

Figure 2.1.s.1: Example Runs Shown for Simon's

```
Enter length of function input [Don't include helper bit in n and ONLY Integer Values Allowed]: 2
Number of iterations:  1
All qubits on Aspen-4-7Q-A: [0, 1, 2, 7, 13, 14, 15]
Only one x st f(x)=1
This is the (randomly chosen) value of a:  [0 1]
Input Possibilities:  [[0, 0], [0, 1], [1, 0], [1, 1]]
Iteration no:  0
Results:
Qubit  0 : [0 0 0 0 0]
Qubit  1 : [1 1 1 1 1]
Time taken by program:  0.16721820831298828
```

Figure 2.1.g.1: Example Runs Shown for Grover's

2.2 Execution time for different U_f

Note: ' * ' next to Time value denotes that sometimes timeout was encountered for corresponding N.

Deutsch-Jozsa Problem

N	Function Type	Time
2	Constant	0.789137601852417
	Constant	0.9590845108032227
	Balanced	0.6435933113098145
	Balanced	0.6755576133728027
3	Constant	1.6737439632415771*
	Constant	1.628145456314087*

	Balanced	1.3247284889221191*
	Balanced	1.2454721927642822*
4	Constant	7.3469460010528564*
	Constant	3.674471139907837*
	Balanced	8.972310890686442*
	Balanced	3.800264596939087*

Table 2.2.dj.1: Observing Effect of Varying U_f on Execution Time for Deutsch-Jozsa

As one can see, regardless of U_f, running Deutsch-Jozsa for the same n takes similar execution time. Balanced functions take slightly less time.

Bernstein-Vazirani Problem

N	Function Type (a,b)	Time
2	A: 01, B: 0	0.7405903339385986
	A: 01, B: 1	0.7191791534423828
	A: 00, B: 0	0.4917905330657959
	A: 10, B: 1	0.6564254760742188
3	A: 000, B: 0	0.45258593559265137*
	A: 100, B: 0	1.3972492218017578*
	A: 010, B: 1	1.9938855171203613*
	A: 101, B: 0	2.3075952529907227*
4	A: 0000, B: 0	0.48230767250061035*
	A: 1110, B: 1	9.030364990234375*

Table 2.2.bv.1: Observing Effect of Varying U_f on Execution Time for Bernstein-Vazirani

As one can see, regardless of U_f, running Bernstein-Vazirani for the same n takes similar execution time. One may notice that peaks may occur.

Simon's Problem

N	Function Type	Time

2	Defined with s= 01	2.1014320850372314
		2.1326191425323486
		2.1115312576293945
3	f1	10.408857345581055*
	f2	10.40829610824585*
	f3	10.35668659210205*
4	Defined with s= 0001	10.802571773529053*
		10.815787076950073*
		10.891235400135128*

Table 2.2.s.1: Observing Effect of Varying U_f on Execution Time for Simon's

As one can see, regardless of U_f, running Simon's for the same n takes similar execution time.

Grover's Problem

N	Function Type	Time
2	Unique x st f(x)=1	0.19576025009155273
	Unique x st f(x)=1	0.15076732635498047
	Multiple x st f(x)=1	0.22897982597351074
	Multiple x st f(x)=1	0.18645834922790527
3	Unique x st f(x)=1	1.004560947418213*
	Unique x st f(x)=1	0.8234810829162598*
	Multiple x st f(x)=1	0.932525634765625*
	Multiple x st f(x)=1	0.6885302066802979*
4	Unique x st f(x)=1	5.443922519683838*
	Unique x st f(x)=1	5.529247999191284*
	Multiple x st f(x)=1	5.940778493881226*
	Multiple x st f(x)=1	4.002136468887329*

Table 2.2.g.1: Observing Effect of Varying U_f on Execution Time for Grover's

As one can see, regardless of U_f, running Grover's for the same n takes similar execution time.

2.3 Execution time as n grows

Note: ' * ' next to Time value denotes that sometimes timeout was encountered for corresponding N. One may also note that we get Timeout results for n >4.

Deutsch-Jozsa Problem

N	Time
1	0.4728856086730957
2	0.7595040798187256
3	1.278507947921753*
4	3.800264596939087*
5	Timeout
6	Timeout

Table 2.3.dj.1: Observing Effect of Varying n on Execution Time for Deutsch-Jozsa

As one can see, on increasing n, execution time for Deutsch-Jozsa algorithm increases. This may approach the exponential trend for higher values of n.

Bernstein-Vazirani Problem

N	Time
1	0.4616396427154541
2	0.7402465343475342
3	0.45258593559265137*
4	0.48230767250061035*
5	Timeout
6	Timeout

Table 2.3.bv.1: Observing Effect of Varying n on Execution Time for Bernstein-Vazirani

As one can see, on increasing n, execution time for Bernstein-Vazirani algorithm increases very slightly. However, one may expect a greater increase in time for higher values of n.

Simon's Problem -

N	Time
2	2.1014320850372314
3 (f1)	10.353229284286499*
3 (f2)	10.35668659210205*
3 (f3)	10.408857345581055*
4	10.802571773529053*

Table 2.3.s.1: Observing Effect of Varying n on Execution Time for Simon's

As one can see, on increasing n, the execution time for Simon's algorithm increases drastically.

Grover's Problem -

N	Time
1	0.1595919132232666
2	0.19948029518127441
3	1.0067853927612305
4	5.594161510467529*
5	Timeout
6	Timeout

Table 2.3.g.1: Observing Effect of Varying n on Execution Time for Grover's

As one can see, on increasing n, execution time for Grover's algorithm increases. This may approach the exponential trend for higher values of n.

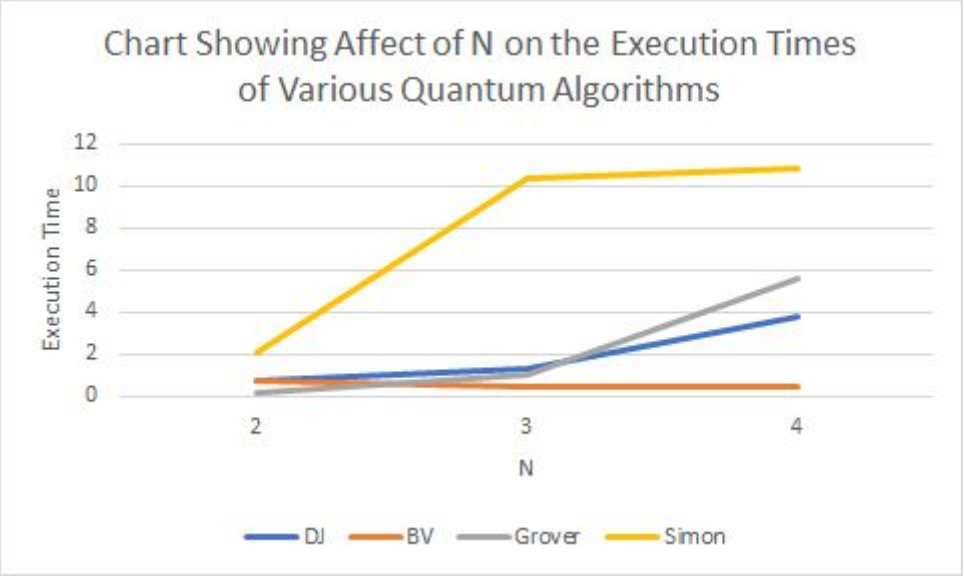

Figure 2.3: Observing Effect of Varying n on Execution Time for Quantum Algorithms.

In the figure, we see that:

1. For DJ and Grover's, with increase in 'n', the execution time starts to increase in what seems like an exponential trend.
2. For Simon's, there is a sharp increase in execution time from n=2 to n=3 but it's not observed in the jump from n=3 to n=4.However, it'll be logical to expect a sharp increase in execution time for higher values on 'n' as there are 2*n qubits involved in Simon's.
3. For BV, we notice there isn't much difference in the execution times for n=2,3 and 4. However, we know from Table 2.2.bv.1 that sometimes for n=4, the execution time can reach 9s and hence, an exponential rise in execution times for higher values of n can be expected.

2.4 Comparison of runs on the quantum computer vs the simulator

When we work on the quantum computer and the simulator, we see that for the same n, execution time is similar. Hence, in the tables below, we compare the results of the simulator and lattice with respect to variation in n. The values in the tables in this section are the same as in 2.3 and are mentioned in column 'Time taken by QCS'. QCS is used to denote the quantum computer and QVM the simulator.

Note:

1. In column Time Taken by QCS (s), ‘ * ’ denotes that for that N sometimes timeout was encountered. For QCS, it was suggested we do not manipulate the timeout as it is not recommended according to Piazza @278 post.
2. In column Time Taken by QCS (s), ‘ ** ’ denotes that for corresponding N, timeout limit was increased and then the value was noted. Without this increase in timeout, the result was usually a timeout.
3. Timeout cases are not considered for the final comparison (even though mentioned) as the timeout for the quantum instance was not manipulated under advisement as mentioned above.

Deutsch-Jozsa Problem

N	Time Taken by QCS (s)	Time Taken by QVM (s)
1	0.4728856086730957	0.07378172874450684
2	0.7595040798187256	0.1047515869140625
3	1.278507947921753*	0.6592450141906738**
4	3.800264596939087*	1.5619616508483887**
5	Timeout	9.98518681526184**
6	Timeout	18.937993049621582**

Table 2.4.dj.1: Comparing Quantum Computer and Quantum Simulator Performance for Deutsch-Jozsa

As one can see, running Deutsch-Jozsa for the same n takes more time on the quantum instance than on the simulator.

Bernstein-Vazirani Problem

N	Time Taken by QCS (s)	Time Taken by QVM (s)
1	0.4616396427154541	0.0801992416381836
2	0.7402465343475342	0.13234996795654297
3	0.45258593559265137*	0.9588367938995361**
4	0.48230767250061035*	6.227278232574463**
5	Timeout	34.44919037818909**
6	Timeout	148.8044970035553**

Table 2.4.bv.1: Comparing Quantum Computer and Quantum Simulator Performance for Bernstein-Vazirani

As one can see, in most cases, running Bernstein-Vazirani for the same n takes more time on the quantum instance than on the simulator.

Simon's Problem -

N	Time Taken by QCS (s)	Time Taken by QVM (s)
2	2.1014320850372314	1.0269525051116943
3 (f1)	10.353229284286499*	53.59334349632263**
3 (f2)	10.35668659210205*	79.37009358406067**
3 (f3)	10.408857345581055*	184.70859694480896**
4	10.802571773529053*	1431**

Table 2.4.s.1: Comparing Quantum Computer and Quantum Simulator Performance for Simons

As one can see, except for the case where n=2 (4 qubits are being used), running Simon's algorithm takes more time on the simulator rather than the quantum instance.

Grover's Problem -

N	Time Taken by QCS (s)	Time Taken by QVM (s)
1	0.1595919132232666	0.123668671
2	0.19948029518127441	0.283242941
3	1.0067853927612305	2.827439785
4	5.594161510467529*	8.887543201446
5	Timeout	10.09999466**
6	Timeout	48.023185014**

Table 2.4.g.1: Comparing Quantum Computer and Quantum Simulator Performance for Grover's

As one can see, except for the case where n=1, running Grover's algorithm takes more time on the simulator rather than the quantum instance.

Part 2. Instruction

The readme file has been attached with the submission files.

Part 3. PyQuil

3.1 Which restrictions on PyQuil programs did you experience when moving from the simulator to the quantum computer?

We were working on our own QMI instance and hence, the issues/restrictions faced may be different from the server jobs allocated for the class.

Reservations: The quantum computer is a limited resource and requires users to reserve/book a lattice. Due to the fact that Rigetti has a lot of users and comparatively lesser number of lattices, it was hard getting convenient reservation times and multiple reservations close to each other. The simulator is available to the user 24*7 and hence this shift in resource availability feels quite demanding.

Ease of debugging: We book a lattice for a slot of time and during that time the main aim is to run our code and check for errors. However, due to the pressure of completing the task within a slot, debugging is not easy. If you book a longer slot and debug for most of the time, you are wasting the resources and money. If you book a shorter slot, while you are saving the resource and some money, you are unable to debug your code at your pace. The finding a balance in this trade-off is quite hard, especially since the simulator is available to the user at all times.

Tracking Intermittent Failures: Also, it did happen that a lot of times the quantum computer gave the correct output, but in a few runs it failed. It was very had to debug such problems because the failure was intermittent. Such intermittent failures were found to occur for each of the algorithms. They were infrequent and due to the restrictive time slots, it took us a long time to completely debug these flaws.

Python functionality and usage: Initially we found it difficult to work with some python functionality like assert statements. This is because the results from the QCS include all of the qubits in the lattice by default and one has to define a structure (say array) for required qubits. In contrast, the simulator works based on qubits defined. Basic differences like this create problems is reusing the PyQuil code from the simulation and using it in the one for the QCS.

Fundamental differences in the working of Simulator and QCS: Since the simulator is for the QCS, one would expect that they would work the same way in some aspects.

However, we found out, the simulator simulates the functionality and not exactly all aspects like structuring and requirements for the quantum computer.

1. Qubits: In the simulator, one can name the qubits as per their liking so for an n-qubit system, usually, by convention, one would prefer to name their qubits by numbers representing 0 to n-1. In contrast to this, the qubit numbering for the quantum computer seems more arbitrary and one needs to modify their code to work with this new system.
2. Gates: For some reason, the instance given by the TA requires the use of primitive gates to develop the custom U_f gate. I think it is due to the native quil compilation required in order to run jobs from multiple teams. When we used our own QMI instance, we did not need to do this as the compilation was directly happening on the quantum computer just like in the simulator. This is a restriction over the creation of gates when compilation methods differ.
3. Timing differences: If one books the lattice, the lattice starts up after 2-3 mins of the start time of the reservation. This wastes some money, especially since the lattice is prompt at shutting down once the slot ends. This problem does not exist while using the simulator.
4. Documentation: When we started using PyQuil and the QVM simulator, we thought the documentation was sparse. However, the documentation for the QCS can be considered worse. There should be documentation eliciting differences in migrating from the simulator to the computer, common errors and occurrences and points to note.