Quantum Linear Solver

By Abhishek Shringi

Agenda

HHL ALGORITHM

PROOF

IMPLEMENTATION OF BASIC GATES IN QISKIT

IMPLEMENTATION OF HHL ALGORITHM IN QISKIT

RUNNING ON QISKIT CLOUD

HHL algorithm

- To solve for x : Ax = b
- Encode A & b as quantum states
- Quantum Phase Estimation
- Controlled Rotation
- Inverse Quantum Phase Estimation
- Measurement

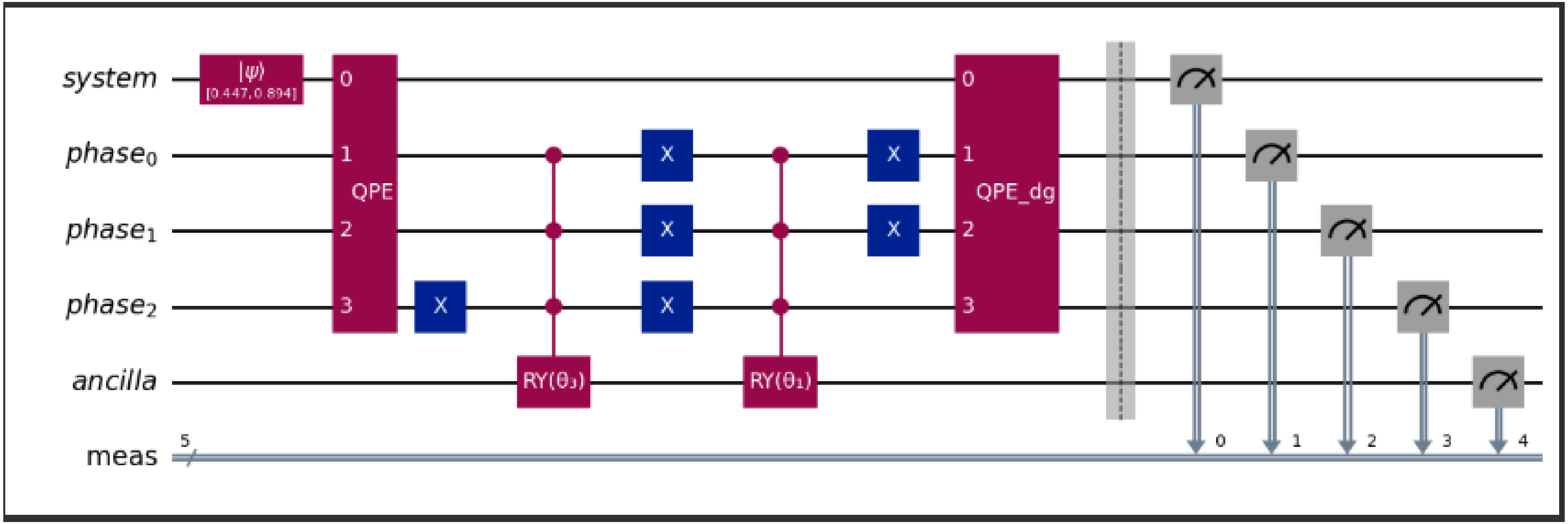
Time complexity for best possible
Classical Algorithm
O(Nsk(1/eps))
but for HHL, it is
O(log(N)(sk)^2/eps)

Mathematical Steps in HHL

- To solve for linear equation $A\,|x\rangle = |b\rangle$; assuming A is Hermitian
- Closed form solution is clearly, $|x\rangle = A^{-1}|b\rangle$
- Since, A is Hermitian
  - $A = \sum_j \lambda_j \, |u_j\rangle\langle u_j|$
  - $A^{-1} = \sum_j (1/\lambda_j) \, |u_j\rangle\langle u_j|$
  - $|b\rangle = \sum_j \beta_j \, |u_j\rangle$
  - $|x\rangle = \sum_j (\beta_j/\lambda_j) \, |u_j\rangle$
- Quantum Phase Estimation
  - U = e^(iAt) ; t = 1
  - $|u_j\rangle \otimes |0\rangle \longrightarrow |u_j\rangle \otimes |\tilde{\lambda}_j\rangle$ (QFT)
  - $|\tilde{\lambda}_j\rangle$ represents the eigenvalue in binary form

Mathematical Steps in HHL

- Controlled Rotation On Ancilla
  - $|u_j\rangle\, |\lambda_j\rangle\, |0\rangle \longrightarrow |u_j\rangle\, |\lambda_j\rangle\, (\sqrt{(1 - (C / \lambda_j)^2)}\, |0\rangle + (C / \lambda_j)\, |1\rangle)$
- Inverse QPE
  - $|u_j\rangle\, |\lambda_j\rangle \longrightarrow |u_j\rangle\, |0\rangle$
- Final State
  - $\sum_j \beta_j\, |u_j\rangle\, (\sqrt{(1 - (C / \lambda_j)^2)}\, |0\rangle + (C / \lambda_j)\, |1\rangle)\, |0\rangle$
- Measurement
  - Measure the state correponding to |1>(ancillary) and |0> in quantum phase

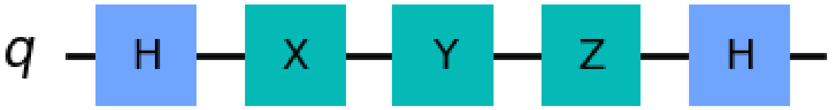

system
phase0
phase1
phase2
ancilla
meas
|ψ⟩
[0.447,0.894]
QPE
X
RY(θ3)
RY(θ1)
QPE_dg
5
0
1
2
3
4

HHL Circuit

Basics of Qiskit Syntax

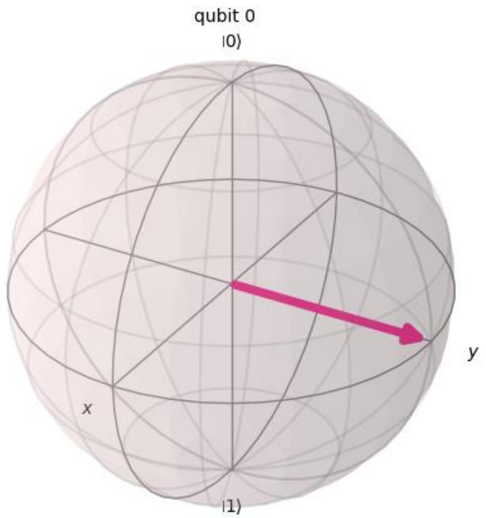
- Clifford Gates(Pauli, H, CNOT) + T Gate -> Universal Gates

```
circuit = QuantumCircuit(1)
circuit.h(0)
circuit.x(0)
circuit.y(0)
circuit.z(0)
circuit.h(0)
circuit.draw(output="mpl")
```



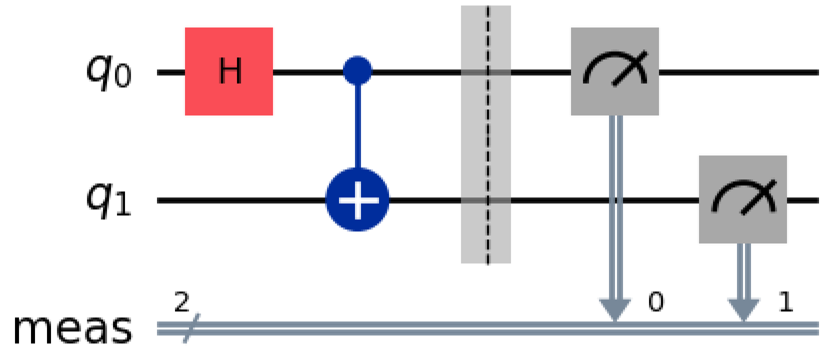

```
v = Statevector(array([1,1.0j])/sqrt(2))
```

$\frac{\sqrt{2}}{2}|0\rangle + \frac{\sqrt{2}i}{2}|1\rangle$

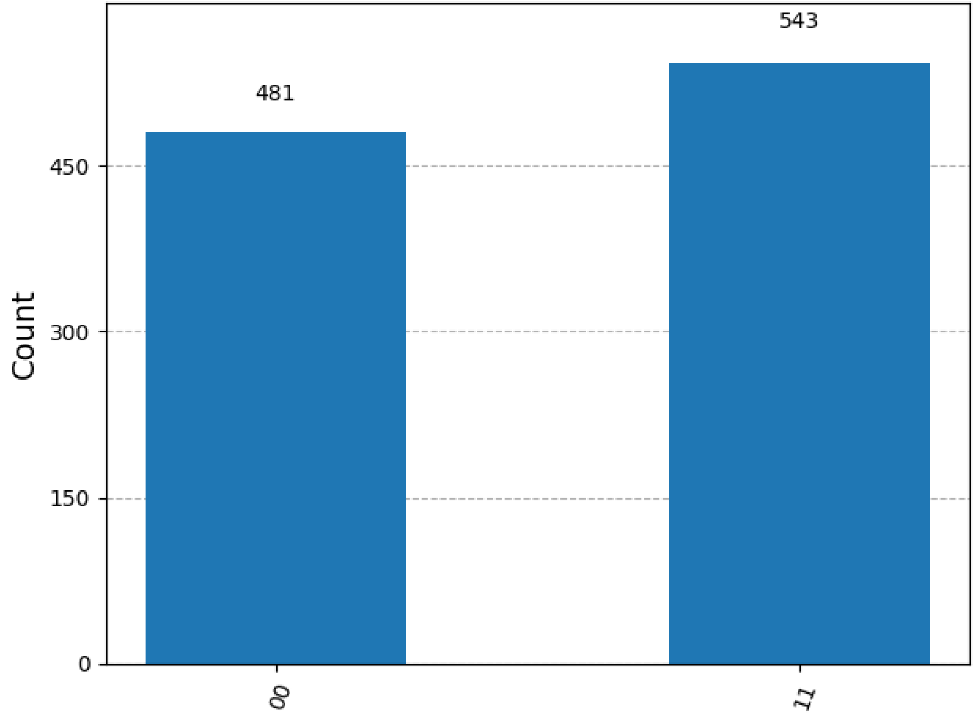


Basics of Qiskit Syntax

- Building Bell State



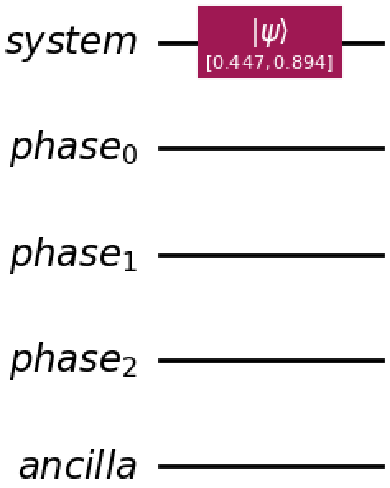

```
qc = QuantumCircuit(2)
qc.h(0)
qc.cx(0, 1)
qc.measure_all()
qc.draw(output="mpl")
simulator_measure = AerSimulator(method="density_matrix")
compiled_circ = transpile(qc, simulator_measure)
result_measure = simulator_measure.run(compiled_circ, shots = 1024).result()
counts = result_measure.get_counts(qc)
plot_histogram(counts)
```



Back To HHL(Example)

- X – 2Y = 2
- -2X + Y = 4
- Solution : [-10/3, -8/3] ~ [-0.7808, -0.6246]
- A = [1 -2 ; -2 1] ; b = [2 ; 4]

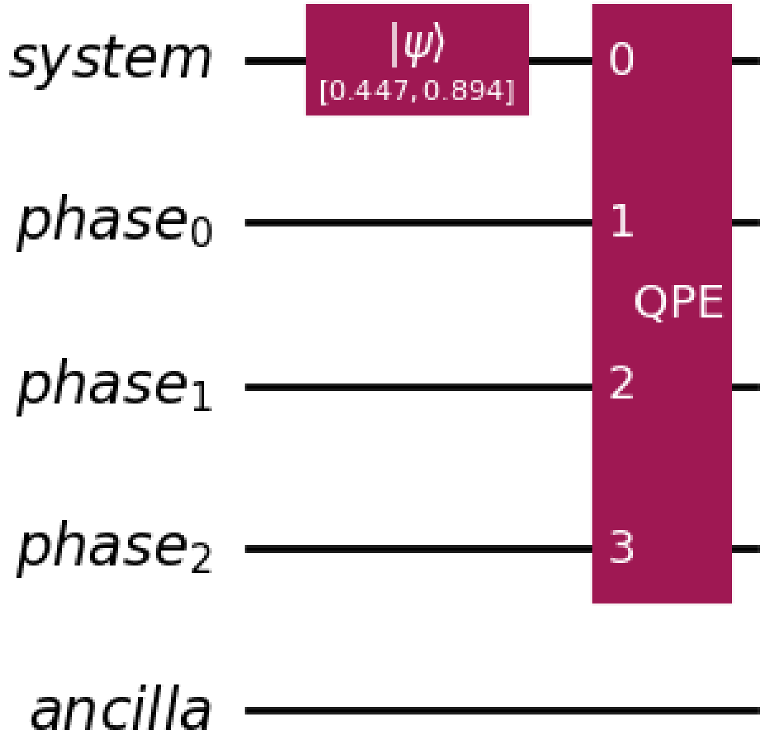
```
A = np.array([[1, -2],
              [-2,  1]])
b = np.array([2, 4])
# Normalize b (note: in the HHL algorithm we encode |b⟩)
A, b_normalized = initialize_problem( A, b )

# ==================================================================
# Create quantum registers:
#   - 1 qubit for the system (to encode |b⟩)
#   - 3 qubits for phase estimation (increased precision)
#   - 1 ancilla qubit for controlled rotation
# ==================================================================
q_system, q_phase, q_anc, qc = initialize_circuit(b_normalized)
qc.draw(output="mpl")
```



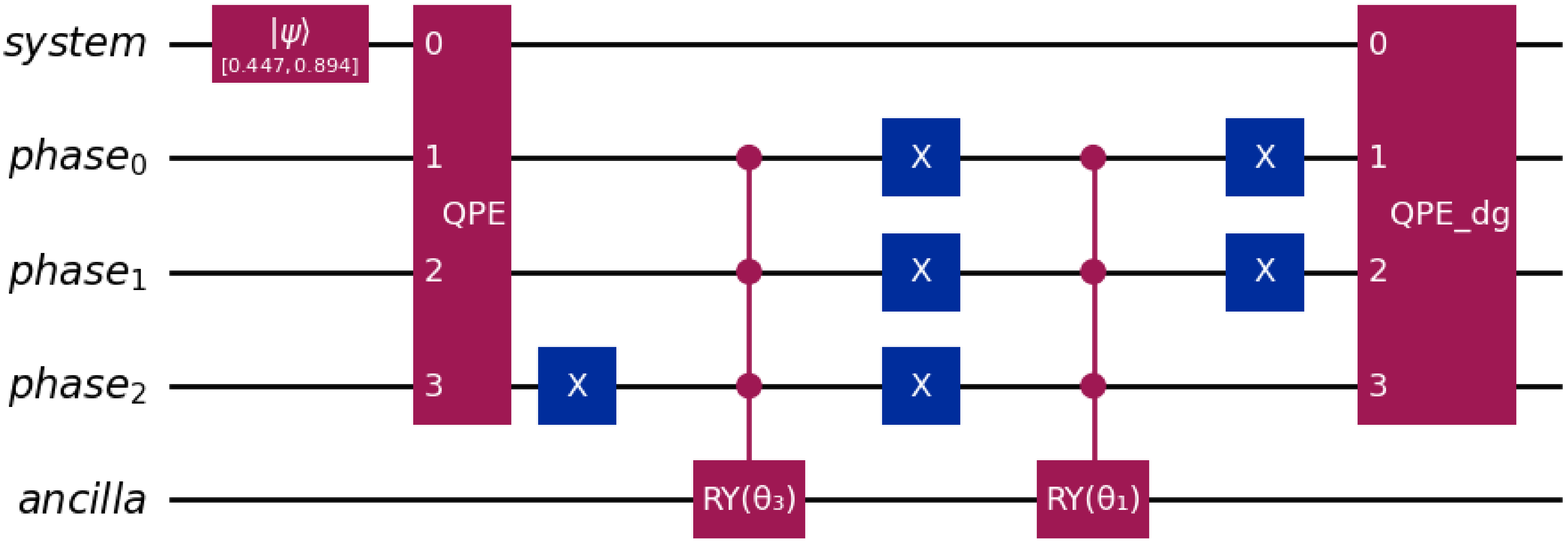

```
Statevector([0.4472136 +0.j, 0.89442719+0.j, 0.        +0.j,
             0.        +0.j, 0.        +0.j, 0.        +0.j,
             0.        +0.j, 0.        +0.j, 0.        +0.j,
             0.        +0.j, 0.        +0.j, 0.        +0.j,
             0.        +0.j, 0.        +0.j, 0.        +0.j,
             0.        +0.j, 0.        +0.j, 0.        +0.j,
             0.        +0.j, 0.        +0.j, 0.        +0.j,
             0.        +0.j, 0.        +0.j, 0.        +0.j,
             0.        +0.j, 0.        +0.j, 0.        +0.j,
             0.        +0.j, 0.        +0.j, 0.        +0.j,
             0.        +0.j, 0.        +0.j],
            dims=(2, 2, 2, 2, 2))
```

Quantum Phase Estimation

```
# Build a QPE subcircuit. Here we use t = 1.
qpe = QuantumCircuit(q_system, q_phase, name="QPE")
qpe.h(q_phase)  # Hadamard on all 3 phase qubits

# We now apply controlled-U^(2^(n-1-j)) for j = 0,1,2 (n=3).
# That is:
#   For j=0 (most significant): exponent = 2^(2)=4
#   For j=1: exponent = 2^(1)=2
#   For j=2 (least significant): exponent = 2^(0)=1
U_4 = expm(1j * 4 * A)  # U^(4)
U_2 = expm(1j * 2 * A)  # U^(2)
U_1 = expm(1j * A)      # U^(1)

cU_4 = UnitaryGate(U_4, label="exp(i4A)").control(1)
cU_2 = UnitaryGate(U_2, label="exp(i2A)").control(1)
cU_1 = UnitaryGate(U_1, label="exp(iA)").control(1)

# Apply controlled unitaries:
# Here we assume q_phase[0] is the most significant qubit.
qpe.append(cU_4, [q_phase[0], q_system[0]])
qpe.append(cU_2, [q_phase[1], q_system[0]])
qpe.append(cU_1, [q_phase[2], q_system[0]])

# Apply the inverse QFT on the 3-qubit phase register.
# We'll use Qiskit's QFT with inverse=True and do_swaps=True.
qft_dagger = QFT(3, inverse=True, do_swaps=True)
qpe.append(qft_dagger.to_instruction(), q_phase[:])
# Append the QPE subcircuit to the main circuit.
qc.append(qpe.to_instruction(), q_system[:] + q_phase[:])
print("\n==== Step 2: Phase Estimation (QPE) with 3 qubits ====")
qc.draw(output='mpl')
```

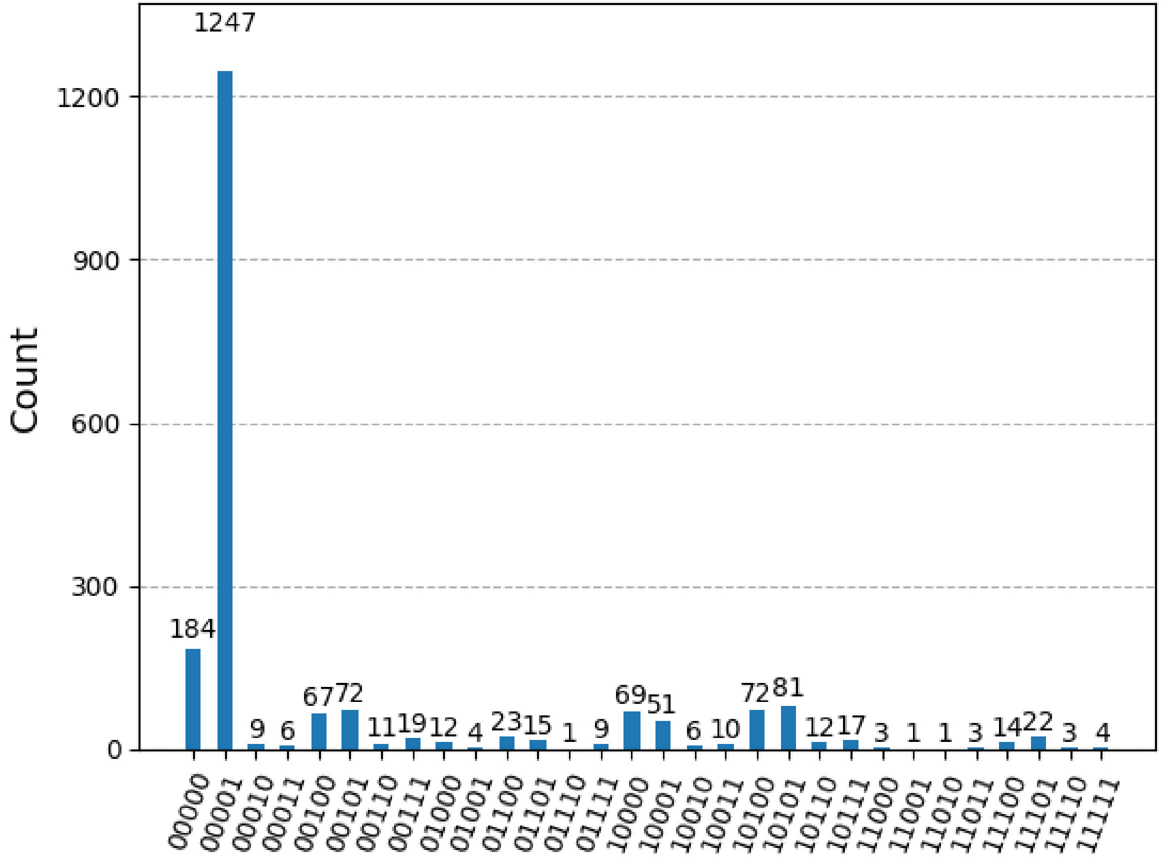


Controlled Rotations & Inverse QPE



Measurements(Simulator)

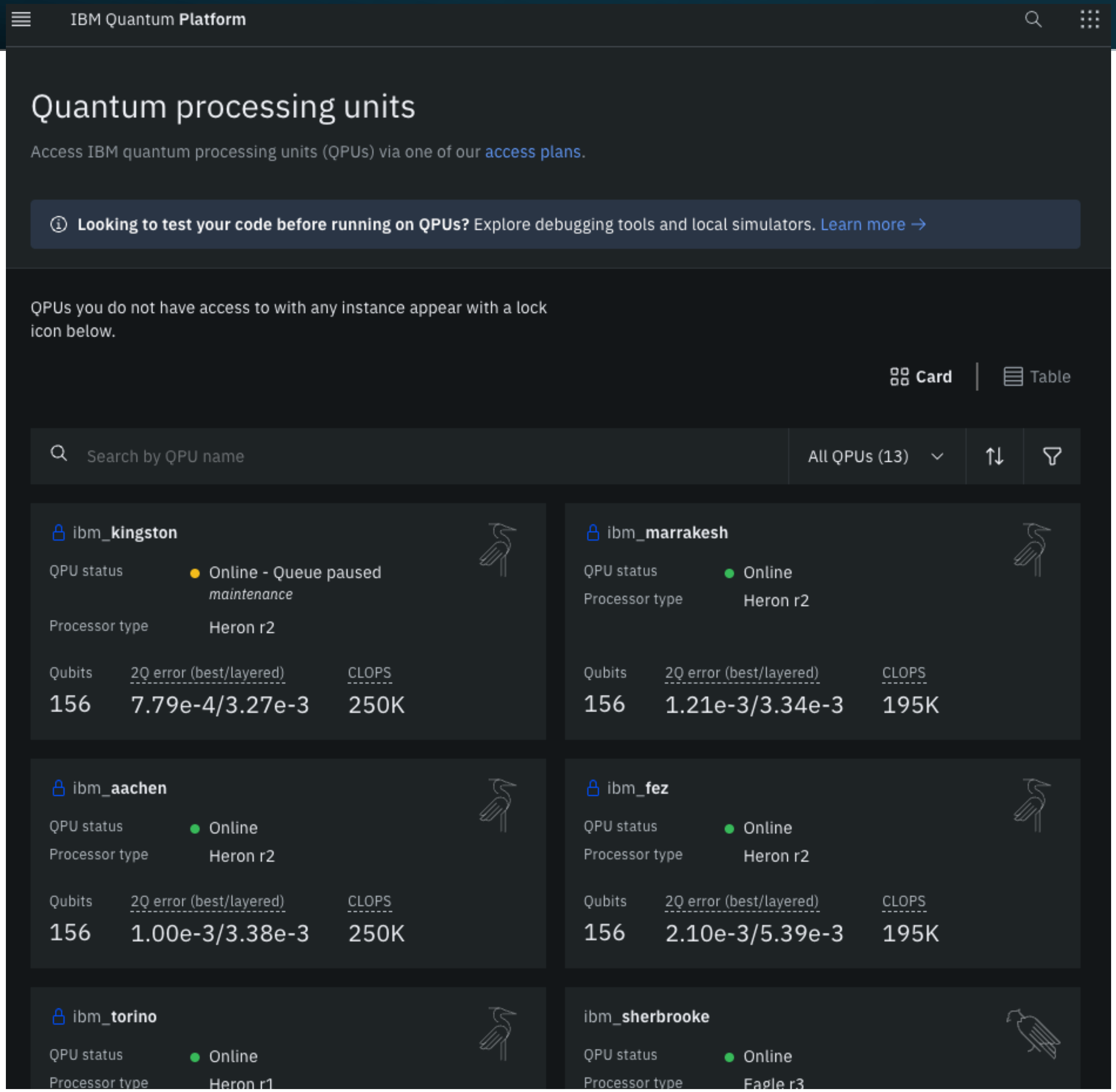
- We measure all the qubits but only consider state corresponding to |000> phase and |1> ancilla qubit



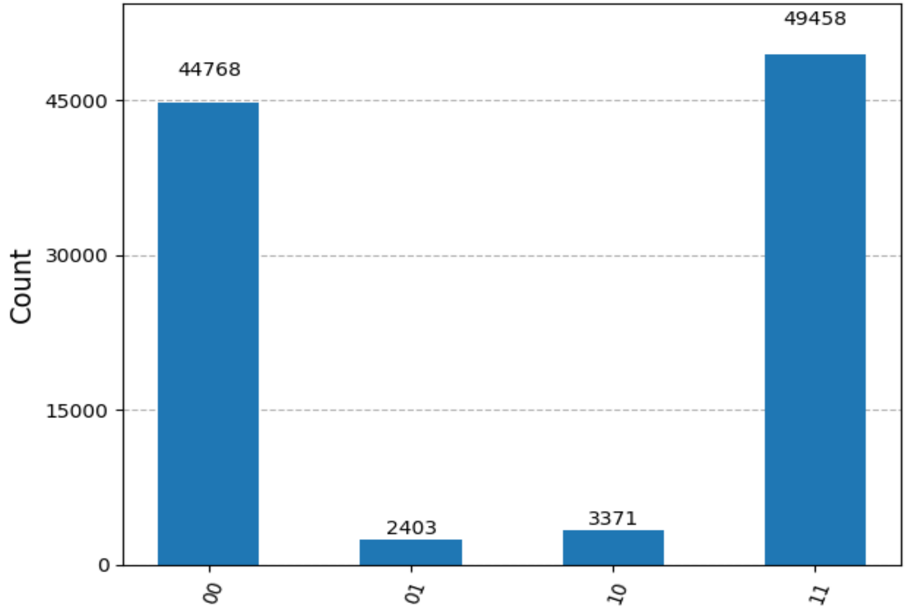

```
Proportional system state |x⟩ (up to normalization):
[0.75828754 0.65192024]
```

```
Original Solution :
array([-0.78086881, -0.62469505])
```

Running On IBM Quantum Hardware




```python
service = QiskitRuntimeService()

bell = QuantumCircuit(2)
bell.h(0)
bell.cx(0, 1)
bell.measure_all()

backend = service.least_busy(operational=True, simulator=False)
pm = generate_preset_pass_manager(backend=backend, optimization_level=1)
isa_circuit = pm.run(bell)

# 3. Execute using the Sampler primitive
sampler = Sampler(mode=backend)
sampler.options.default_shots = 100000  # Options can be set using auto-complete.
job = sampler.run([isa_circuit])
print(f"Job ID is {job.job_id()}")
pub_result = job.result()[0]
print(f"Counts for the meas output register: {pub_result.data.meas.get_counts()}")
```

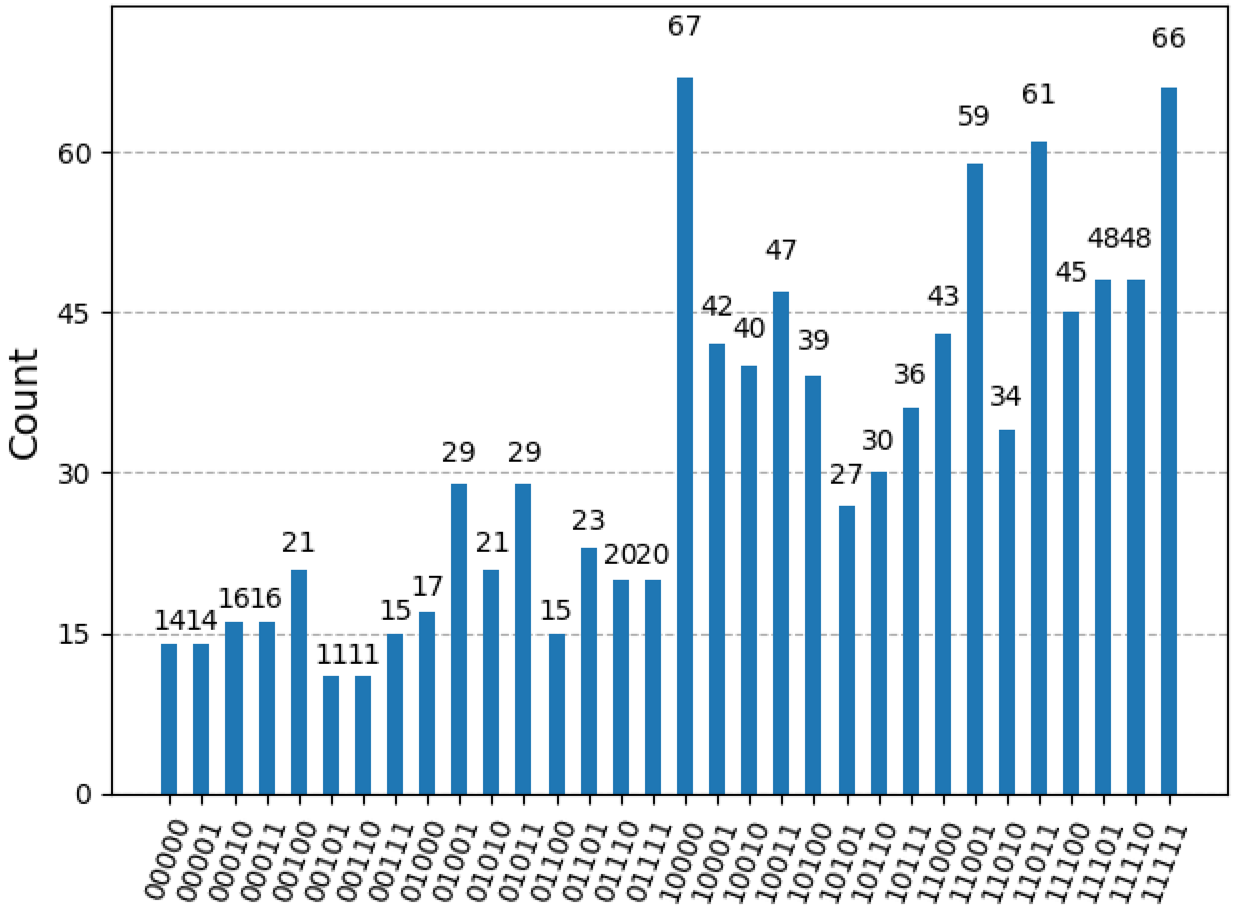


Running HHL

Excecuted on IBM_SHERBROOKE for 1024 shots

```
Proportional system state
[0.7840146  0.62074238]
```

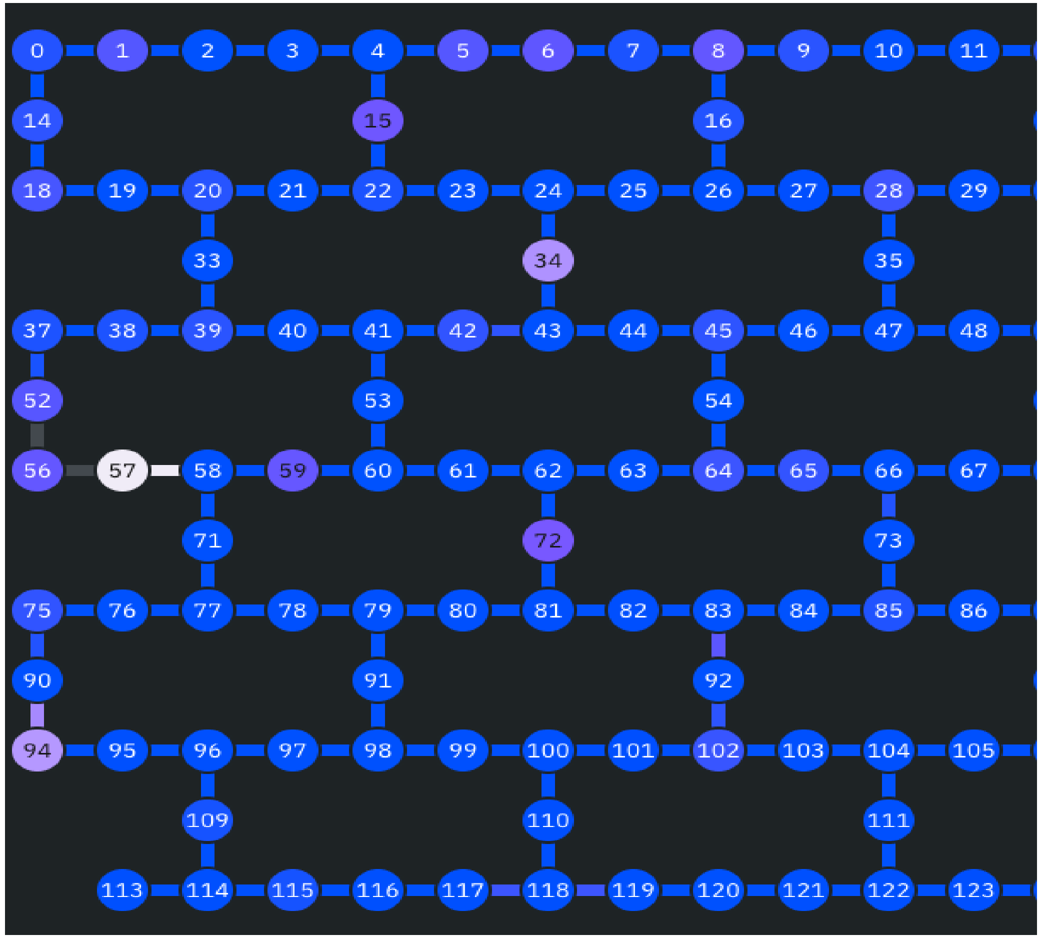
```
Original Solution :
array([-0.78086881, -0.62469505])
```



Limitations/Scope Of Improvement

- Quantum Error Correction/Mitigation : Of Course!

- Circuit Depth
  - The naïve implementation of algorithms can lead to large circuit depths which could lead to inefficiencies especially in a limited qubit scenario where connectivity becomes a constraint



References

- Zaman_2023, title={A Step-by-Step HHL Algorithm Walkthrough to Enhance Understanding of Critical Quantum Computing Concepts}
- Harrow_2009, title={Quantum Algorithm for Linear Systems of Equations},