SIMULACIÓ DEUTSCH

En aquest darrer exercici del curs, es simularà i s'executarà l'algorisme de *Deutsch* que s'ha treballat a classe en un ordinador quàntic real utilitzant l'entorn Qiskit de IBM. Aquest entorn, ens permet dissenyar, simular i executar diferents circuits quàntics usant o bé l'entorn gràfic, *Quantum Composer* o bé amb un notebook de `Python`, anomenat *Quantum Lab*.

L'execució es realitzarà en un ordinador quàntic real, de forma gratuïta i ràpida, però limitada a un màxim de 5 qubits, que serà més que suficient per a la nostra tasca.

Organitzarem l'exercici com es mostra sota el paràgraf, seguint l'enunciat penjat a *Atenea* i amb els apartats que es veuen a la taula de continguts següent:

TAULA DE CONTINGUTS

PREPARACIÓ DE L'ENTORN 1
CIRCUIT DE PARELLS ENTRELLAÇATS 2
NOTEBOOK DE PARELLS ENTRELLAÇATS 3
SIMULACIÓ I MUNTATGE DEL CIRCUIT DE DEUTSCH 7
- **MUNTATGE DEL CIRCUIT DE DEUTSCH 7**
- **NOTEBOOK DEL CIRCUIT DE DEUTSCH 9**
- **ANÀLISI I VERIFICACIÓ DELS RESULTATS 12**

EXECUCIÓ DE L' ALGORISME DE DEUTSCH EN UN ORDINADOR QUÀNTIC REAL 13
- **COM CONSULTAR ELS ORDINADORS QUÀNTICS 13**
- **NOTEBOOK AMB L'ALGORISME ENVIAT A L'ORDINADOR QUÀNTIC 14**
- **RESULTATS OBTINGUTS 16**

EXEMPLES GUIATS 19

PREPARACIÓ DE L'ENTORN

Per a dur a terme l'exercici, primer de tot cal registrar-se o bé loggejar-se a la pàgina de https://quantum-computing.ibm.com/. És recomanable fer-ho mitjançant **IBMid** i evitar fer-ho amb una compte **.edu**, ja que en el meu cas em va portar problemes a l'hora de loggejar-me un cop ja creat el nou compte.

Un cop ens hem loggejat i hem entrat amb el nostre usuari, podrem veure el panell de benvinguda, com es mostra a continuació:

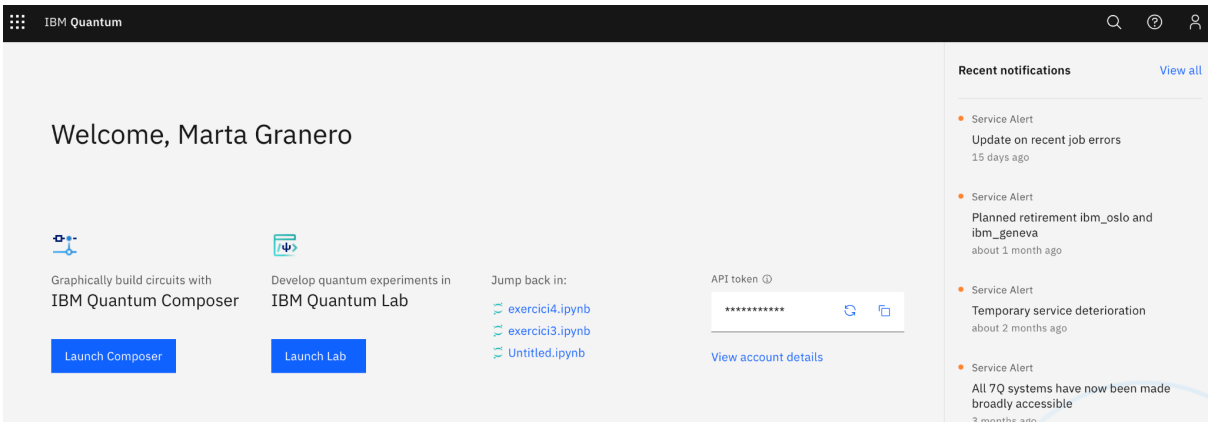


Figura 1: Panell de benvinguda

CIRCUIT DE PARELLS ENTRELLAÇATS

Un cop ja som al panell, procedirem a fer el nostre primer circuit, i ho farem amb l'entorn visual del Quantum Composer. Per fer-ho, des del panell farem clic a **Launch Composer.**

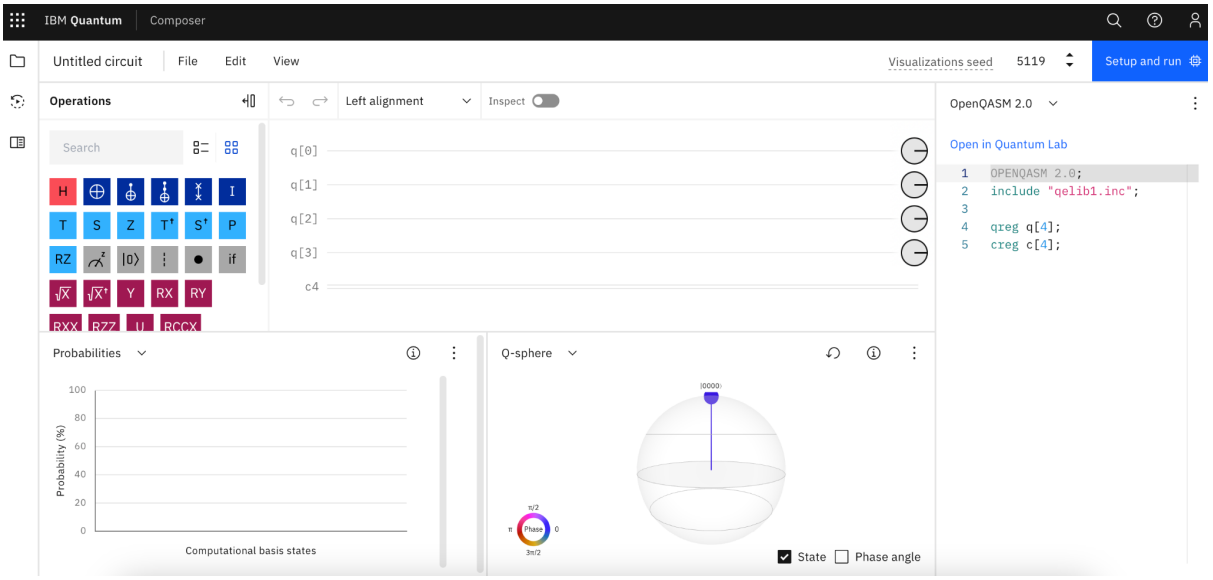


Figura 2: Entorn visual del Quantum Composer

Aquest entorn ens ofereix una representació visual per crear un un circuit i ens posa a la nostra disposició un ventall d'operadors unitaris per construir el circuit que vulguem.

En el nostre cas, el circuit que muntarem produirà parells entrelaçats[1] mitjançant una porta *Hadamard* i una porta *C-NOT*. Per dur-ho a terme el muntatge, arrossegarem

[1] https://quantum-computing.ibm.com/composer/docs/iqx/guide/entanglement

les portes quàntiques fins al les línies de qubits, eliminarem els qubit q[2] i q[3] i afegirem les icones de mesura pels dos qubits restants.

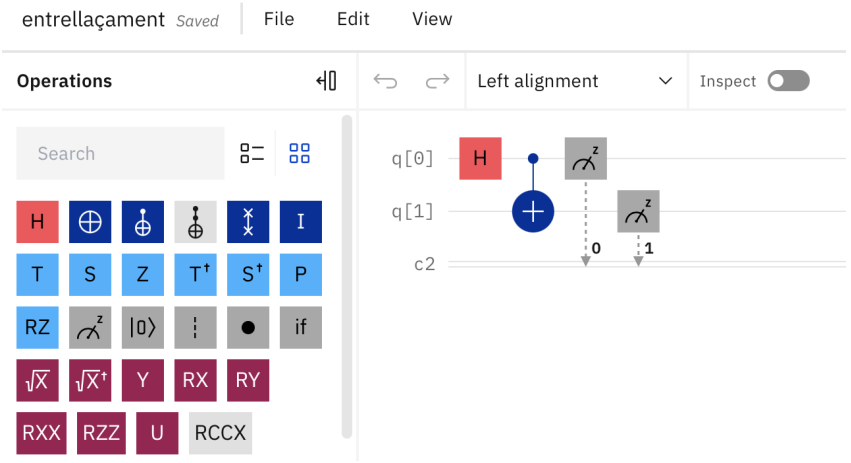


Figura 3: Circuit que crea parells entrellaçats

Com podem veure, el nostre circuit crea doncs parells entrellaçats, on tenim una probabilitat del 51.5625% de mesurar l'estat 00 i una del 48.4375% de mesurar 11, però una probabilitat de 0% de mesurar els estats 01 i 10.

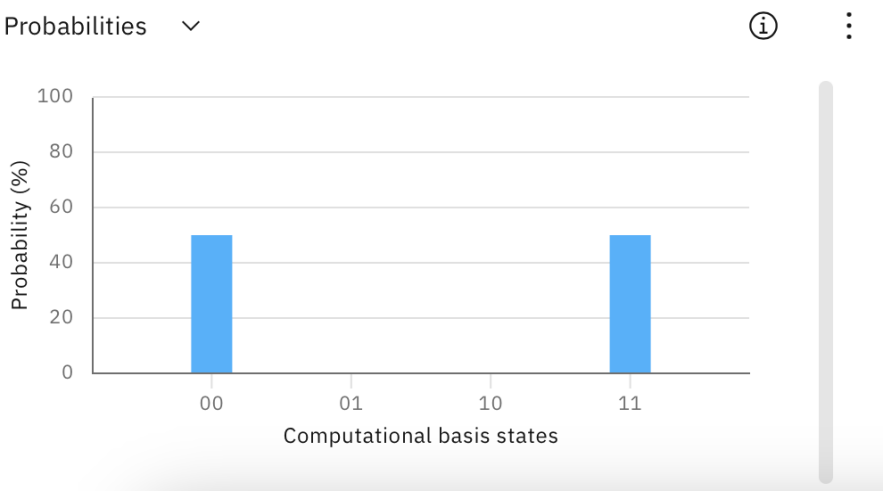


Figura 4: Probabilitats de mesurar els estats 00 i 11

És interessant notar que segons la seed de visualització[2], les probabilitats seran diferents. La que s'ha usat i es veu a la Figura 4 és la seed 5119.

NOTEBOOK DE PARELLS ENTRELLAÇATS

Ara, durem a terme el mateix propòsit de construir parells entrellaçats però fent ús d'un notebook de Python. Per fer-ho, caldrà anar altre cop al panell de benvinguda i clicar sobre **Launch Lab** (veure la Figura 1) per entrar a l'entorn de Quantum Lab.

[2] https://quantum-computing.ibm.com/composer/docs/iqx/visualizations#probabilities-view

Un cop a dins, executarem el servidor, fent clic sobre **Launch Server**, aquí, en alguns casos cal esperar a tenir el servir preparat fins que automàticament es llença el servei.

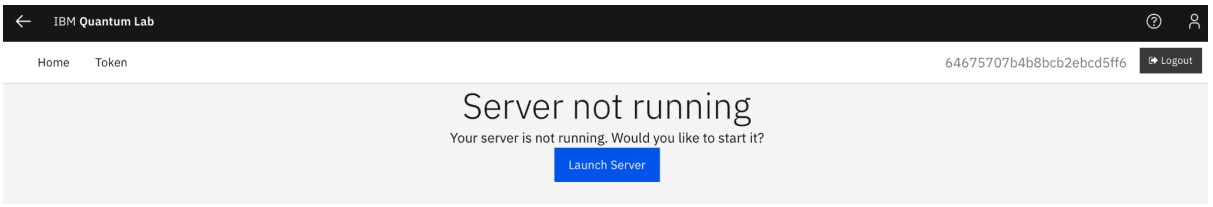


Figura 5: Executem el servidor per entrar a Quantum Lab

Un cop es llença el servei, podrem veure el *Launcher*. Aquí se'ns mostra un ventall d'opcions, nosaltres farem servir la primera opció que s'anomena: `Python 3` amb el kernel (*ipykernel)* i que té el logo de *Qiskit.*

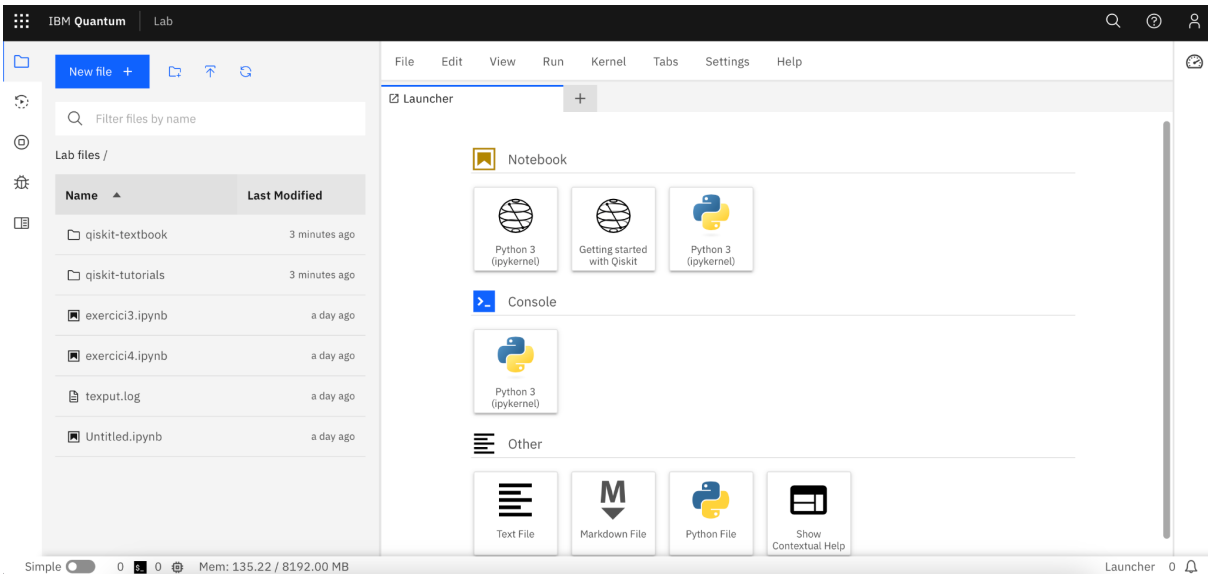


Figura 6: Launcher de Quantum Lab

Un cop obert el notebook, copiarem les instruccions que es troben a l'enllaç, https://quantum-computing.ibm.com/lab/docs/iql/first-circuit, les quals implementen exactament el mateix circuit de l'apartat anterior.

Per fer-ho s'hauran de copiar les línies que apareixen als apartats *Build, Execute, Analyze, Visualize* de l'enllaç que hem obert. Prèviament però, haurem d'importar els packages, mòduls de Qiskit i funcions per poder executar-ho correctament, ja que els imports que es carreguen per defecte no inclouen algunes llibreries que ens permeten executar el codi que copiem!

```
# Importing standard Qiskit modules and configuring account
```

```
from qiskit import QuantumCircuit, execute, Aer, IBMQ
from qiskit.compiler import transpile, assemble
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from ibm_quantum_widgets import *

# Loading your IBM Q account(s)
provider = IBMQ.load_account()
```

Figura 7: Bloc de codi que s'ha d'afegir a la notebook

A continuació es mostra la resta del codi:

```
# Build
#------
# Create a Quantum Circuit acting on the q register
circuit = QuantumCircuit(2, 2)
# Add a H gate on qubit 0
circuit.h(0)
# Add a CX (CNOT) gate on control qubit 0 and target qubit 1
circuit.cx(0, 1)
# Map the quantum measurement to the classical bits
circuit.measure([0,1], [0,1])
# END

# Execute
#--------
# Use Aer's qasm_simulator
simulator = Aer.get_backend('qasm_simulator')
# Execute the circuit on the qasm simulator
job = execute(circuit, simulator, shots=1000)
# Grab results from the job
result = job.result()
# Return counts
counts = result.get_counts(circuit)
print("\nTotal count for 00 and 11 are:",counts)
# END

# Visualize
#----------
# Import draw_circuit, then use it to draw the circuit
from ibm_quantum_widgets import draw_circuit
```

```
draw_circuit(circuit)

# Analyze
#--------
# Plot a histogram
plot_histogram(counts)
# END
```

Figura 8: Resta de codi que permet crear parells entrellaçats

El codi copiat té diverses seccions, cadascuna idealment hauria d'anar amb una cel·la de la notebook per poder visualitzar els resultats de forma parcial.

Si analitzem què fa cada secció, començant des de la *Build, Execute, Visualize* i acabant amb la de *Analyze*, podem veure que les diferents seccions ens permeten obtenir:

- Build: Construcció del circuit quàntic que crea el parell entrellaçat. Ho fem amb una porta Hadamard, una porta CNOT i els aparells de mesura en cadascun dels qubits.

- Execute: Ens permet executar el nostre circuit **1000** vegades mitjançant la funció execute i fent servir el simulador *qasm_simulator.*

  A cada execució, aquest circuit donarà com a resultat o bé l'estat 00 o bé el 11. Un cop tenim l'objecte que ens guarda el resultat, en aquest cas la variable **job**, podem accedir al seu mètode de get_counts per saber el nombre de cops que ha sortit l'estat 00 i l'estat 11. Com s'esperava, obtenim que el aproximadament el 50% del temps surt l'estat 00 i l'altre l'estat 11. No surt **50%/50%** ja que aquest simulador no modela el soroll i qualsevol desviació del 50% es deu a la petita mida de la mostra.

  Hem obtingut en el nostre cas, el resultat de:

  ```
  Total count for 00 and 11 are: {'00': 498, '11': 502}
  ```

- Visualize: Podem visualitzar el circuit amb l'aparença i la sensació de l'entorn visual del Quantum Composer amb la funció `draw_circuit(circuit)` i important la llibreria `from ibm_quantum_widgets import draw_circuit`

Obtenim com a resultat el circuit:

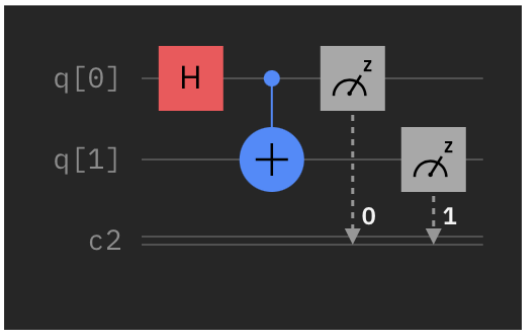


Figura 9: Circuit que hem obtingut a l'executar la funció draw_circuit

- Analize: Amb la funció de plot_histogram, podrem observar les freqüències relatives d'observar els estats 00 i 11 que hem obtingut fent l'execució.

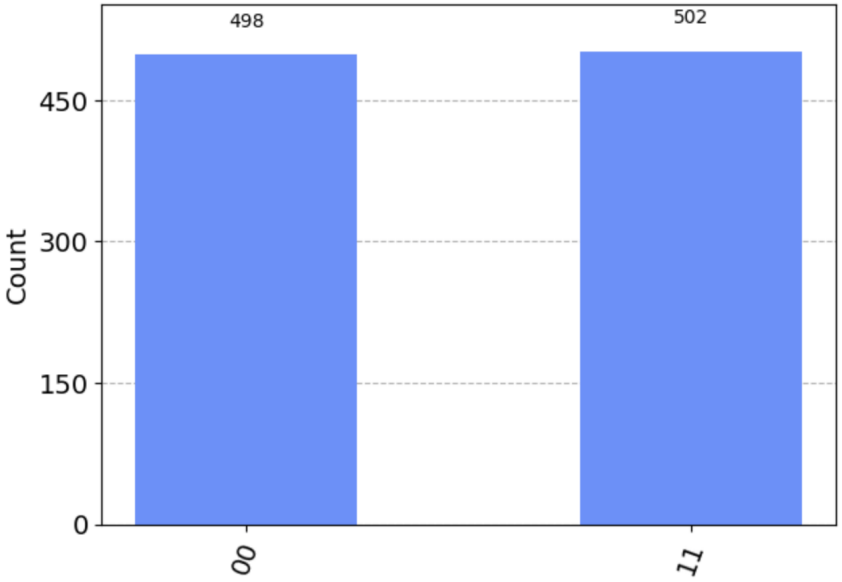


Figura 10: Histograma amb el nombre de vegades que surt cada estat

Veiem doncs que obtenim un resultat semblant respecte a la secció anterior, ara però, és més probable, obtenir l'estat 11 que l'estat 00, respectivament amb un 50.2% en contra de 49.8%. Notem que a cada execució, obtindrem uns resultats diferents, tal i com passava en l'apartat anterior.

SIMULACIÓ I MUNTATGE DEL CIRCUIT DE DEUTSCH

MUNTATGE DEL CIRCUIT DE DEUTSCH

Seguint amb el muntatge i disseny de circuits quàntics, ara ho farem per l'algorisme de Deutsch. El circuit sencer que implementa l'algorisme i que s'ha creat amb el visualitzador de Quantum Composer és el següent:

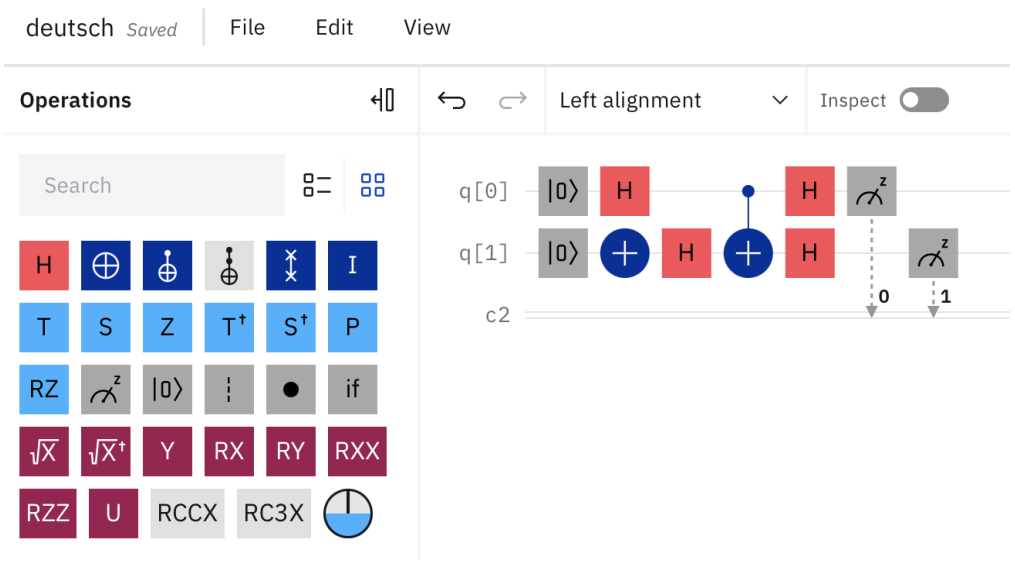


Figura 11: Circuit que implementa l'algorisme de Deutsch

Per construir el circuit de Deutsch per a una funció balancejada del tipus f(0) = 0 i f(1) = 1 ho hem fet de la següent manera:

1. Inicialitzem els dos qubits en els estats |0⟩. On els anomenem com: qubit 0 (q0) i qubit 1 (q1).
2. Apliquem una porta Hadamard (H) al primer qubit (q0).
3. Apliquem una porta NOT al segon qubit (q1).
4. Apliquem una porta Hadamard (H) al segon qubit (q1).
5. Apliquem una porta CNOT controlada pel primer qubit (q0) i target el segon qubit (q1).
6. Mesurem els dos qubits.

També, podem veure la probabilitat que surti cada estat:

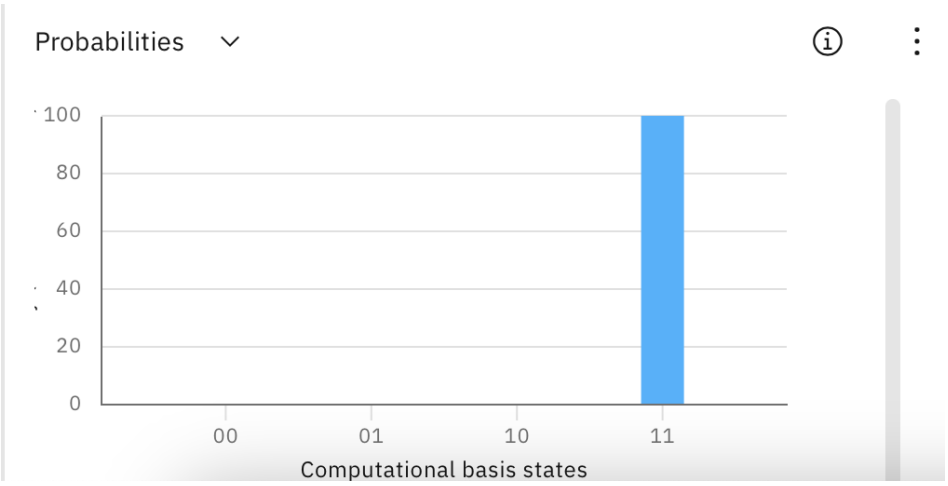


Figura 12: Probabilitat que surti cadascun dels estats quàntics

Cal tenir en compte però que el segon qubit l'hem d'inicialitzar a |1⟩, malgrat per defecte tots els qubits s'inicialitzen a |0⟩. Hem aplicat una porta NOT a aquest qubit, tal i com hem vist a la Figura 11.

NOTEBOOK DEL CIRCUIT DE DEUTSCH

Podem escriure també un notebook que implementa el mateix circuit creat amb el Composer.

El codi és el següent:

```
# Importing standard Qiskit modules and configuring account
from qiskit import QuantumCircuit, execute, Aer, IBMQ
from qiskit.compiler import transpile, assemble
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from ibm_quantum_widgets import *
# Loading your IBM Q account(s)
provider = IBMQ.load_account()

# Build
#------
# Create a Quantum Circuit acting on the q register
circuit = QuantumCircuit(2, 2)
# Add a H gate on qubit 0
circuit.h(0)
# Add a X gate on qubit 1
circuit.x(1)
# Add a H gate on qubit 1
circuit.h(1)
# Add a CX (CNOT) gate on control qubit 0 and target qubit 1
circuit.cx(0, 1)
# Add a H gate on qubit 0 and 1
circuit.h(0)
circuit.h(1)
# Map the quantum measurement to the classical bits
circuit.measure([0,1], [0,1])
# END

# Execute
#--------
# Use Aer's qasm_simulator
simulator = Aer.get_backend('qasm_simulator')
```

```
# Execute the circuit on the qasm simulator
job = execute(circuit, simulator, shots=1000)
# Grab results from the job
result = job.result()
# Return counts
counts = result.get_counts(circuit)
print("\nTotal count for 00 and 11 are:",counts)
# END

# Visualize
#----------
# Import draw_circuit, then use it to draw the circuit
from ibm_quantum_widgets import draw_circuit
draw_circuit(circuit)
# Analyze
#--------
# Plot a histogram
plot_histogram(counts)
# END
```

Figura 13: Bloc de codi que implementa el circuit de Deutsch

Notar però que la única part que cambia respecte el codi de l'apartat anterior és la secció de ***Build***

Executant aquest codi hem obtingut els circuit següent:

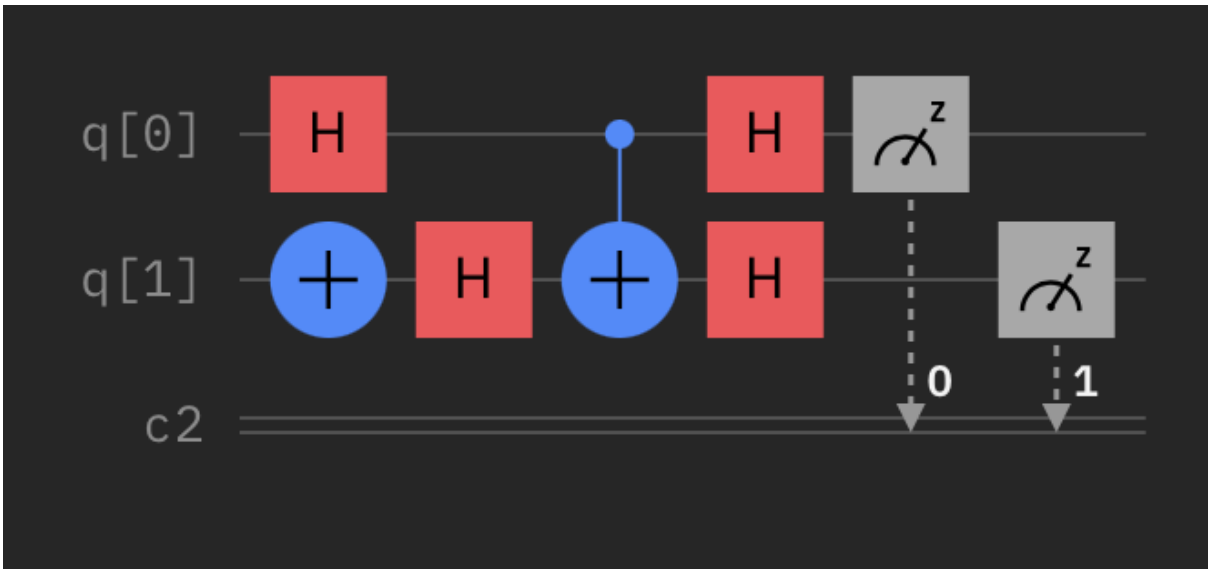


Figura 14: Circuit de Deutsch per a dos qubits

A més a més, entre les 1000 execucions, en totes i cadascuna d'elles ha sortit l'estat 11. On per tant, veiem que el 100% del temps obtenim l'estat 11. On queda clar que la probabilitat d'obtenir l'estat 11 és del 100%.

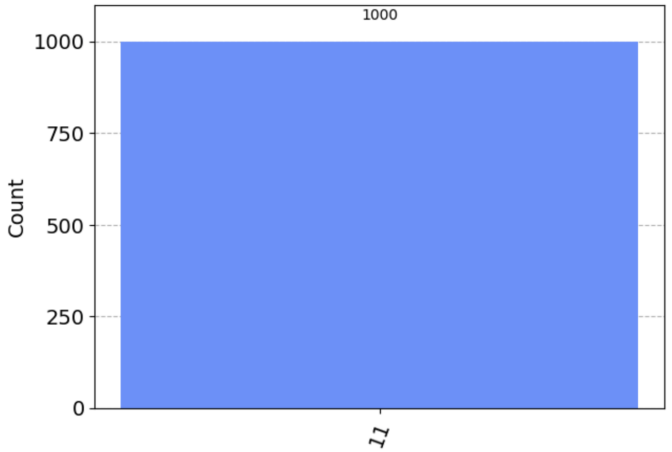
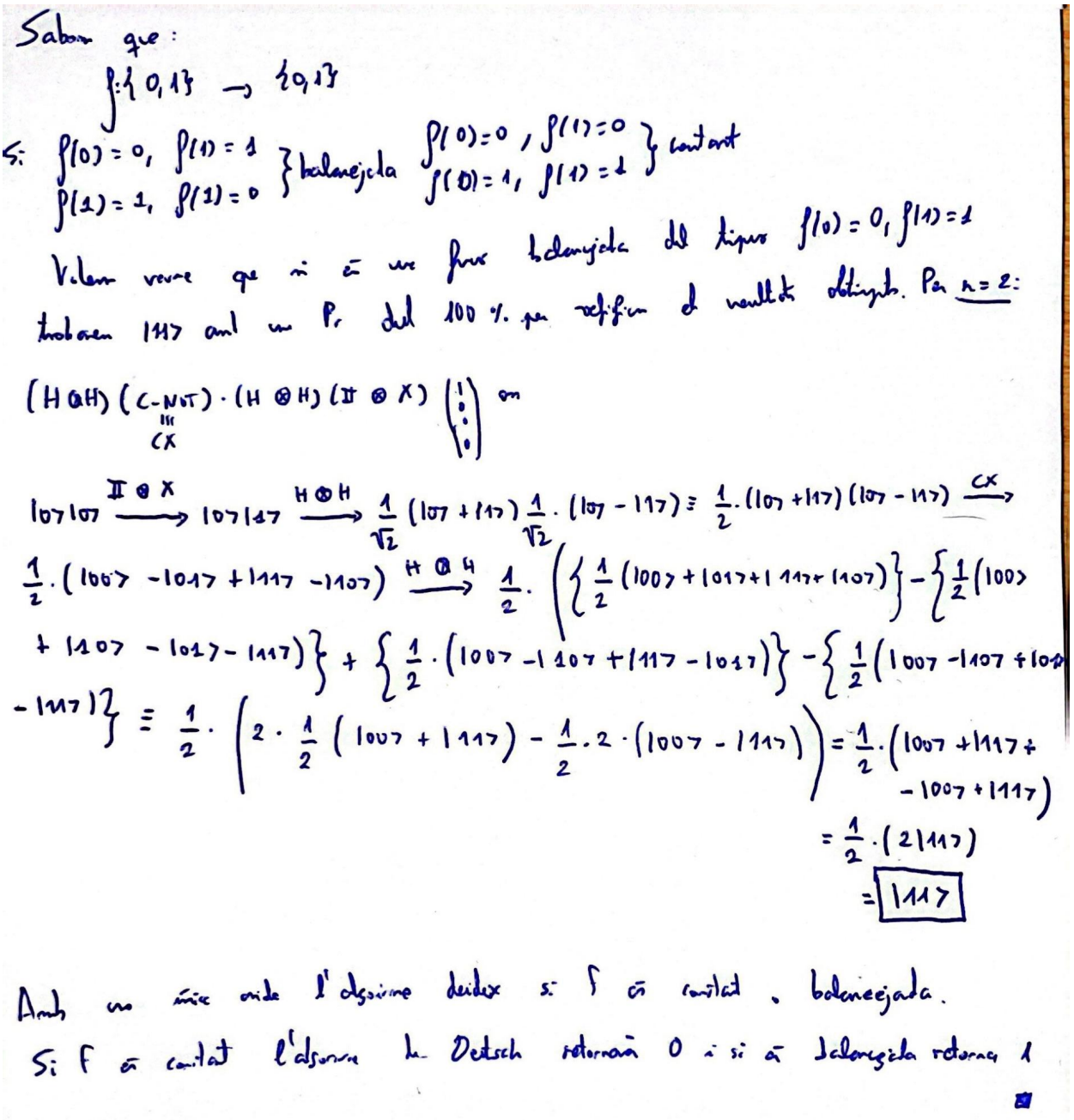


Figura 15: Histograma on veiem que en totes les execucions surt l'estat 11

Cal notar que la implementació del circuit de Deutsch per a la funció equilibrada del tipus f(0) = 0, f(1) = 1 és molt similar al que hem fet en l'apartat anterior.

ANÀLISI I VERIFICACIÓ DELS RESULTATS

Ara que ja hem executat el codi, per saber si realment el resultat que ens retorna és correcte, ens cal verificar-ho. Veurem que, aquest resultat és correcte, ja que si fem el càlcul pas a pas, aplicant el producte de matrius tindrem:



EXECUCIÓ DE L' ALGORISME DE DEUTSCH EN UN ORDINADOR QUÀNTIC REAL

Ara finalment, ens queda executar l'algorisme de Deutsch que hem elaborat en un ordinador quàntic real. Per executar el circuit creat i poder enviar-lo en un dels que ens ofereix IBM s'han segut les instruccions detallades de l'enllaç: https://qiskit.org/textbook/ch-algorithms/deutsch-jozsa.html, concretament les de l'apartat *Experiment with real devices*.

COM CONSULTAR ELS ORDINADORS QUÀNTICS

Per poder consultar quins són els backends i simladors que ens ofereix IBM, i veure quins ordinadors quàntics té disponibles, podem executar el fragment de codi següent(després d'importar les llibreries):

```
IBMQ.load_account() # Load account from disk
IBMQ.providers()    # Ens mostra la llista de tots els providers
provider = IBMQ.get_provider(hub='ibm-q')
provider.backends() # Ens mostra la llista de backends disponibles
```

Si volem consultar diverses característiques de l'ordinador quàntic(jobs que té pendents, característiques dels qubits, entre d'altres, …), podem executar el següent codi i substituir `'ibmq_manila'` pel backend que ens interessi més.

Si fem una ullada a la llista dels backends es pot veure que alguns ordinadors quàntics disponibles compten amb més de 5 qubits, com és el cas de l'ordinador quàntic de Nairobi.

```
backend = provider.get_backend('ibmq_manila')
backend
```

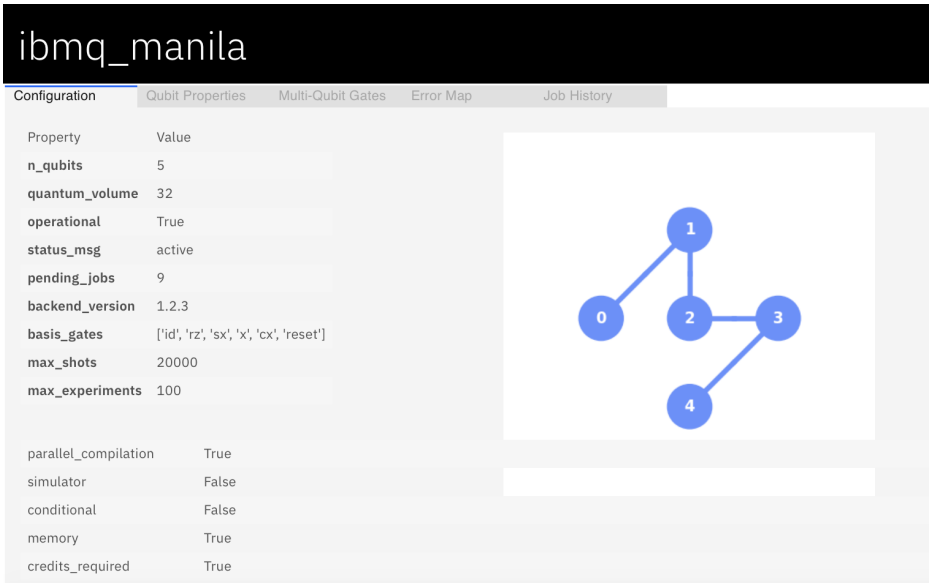


Figura 16: Característiques de l'ordinador quàntic de Manila

NOTEBOOK AMB L'ALGORISME ENVIAT A L'ORDINADOR QUÀNTIC

El codi que hem enviat a l'ordinador quàntic de Manila té un paregut gairebé idèntic amb el codi de l'apartat anterior, simplement hem afegit:

- n = 2 (# de qubits que ens interessen)
- La funció ***least_busy*** per obtenir el backend amb menys jobs pendents per poder afegir el nostre a la cua. En el nostre cas ha set l'ordinador `'ibmq_manila'`
- Hem canviat `circuit = QuantumCircuit(n, n)` i enlloc de tenir (2,2) ho hem generalitzat amb la variable n.
- Hem canviat la secció de *Execute* i hem afegit les línies:

```
transpiled_circuit = transpile(circuit, backend, optimization_level=3)
job = backend.run(transpiled_circuit)
job_monitor(job, interval=2) #Monitoritzem el nostre job
```

El codi és el següent:

```
from qiskit import QuantumCircuit, execute, Aer, IBMQ
from qiskit.compiler import transpile, assemble
from qiskit.providers.ibmq import least_busy
from qiskit.tools.jupyter import *
from qiskit.visualization import *
```

```
from ibm_quantum_widgets import *
from qiskit.tools.monitor import job_monitor
from qiskit.visualization import plot_histogram

# Nombre de qubits que ens interessa
n = 2

# Load our saved IBMQ accounts and get the least busy backend
device with greater than or equal to (n+1) qubits
IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
backend = least_busy(provider.backends(filters=lambda x:
x.configuration().n_qubits >= (n+1) and
                                       not
x.configuration().simulator and x.status().operational==True))
print("least busy backend: ", backend)

# Build
#------
# Create a Quantum Circuit acting on the q register
circuit = QuantumCircuit(n, n)
# Add a H gate on qubit 0
circuit.h(0)
# Add a X gate on qubit 1
circuit.x(1)
# Add a H gate on qubit 1
circuit.h(1)
# Add a CX (CNOT) gate on control qubit 0 and target qubit 1
circuit.cx(0, 1)
# Add a H gate on qubit 0 and 1
circuit.h(0)
circuit.h(1)
# Map the quantum measurement to the classical bits
circuit.measure([0,1], [0,1])

# Execute
#--------
transpiled_circuit = transpile(circuit, backend,
optimization_level=3)
job = backend.run(transpiled_circuit)
job_monitor(job, interval=2)
```

```
# Execute the circuit on the qasm simulator
job = execute(circuit, backend, shots=1000)
# Grab results from the job
result = job.result()
# Return counts
counts = result.get_counts(circuit)
print("\nTotal count for 0 and 1 are:",counts)
# END

# Visualize
#----------
# Import draw_circuit, then use it to draw the circuit
from ibm_quantum_widgets import draw_circuit
draw_circuit(circuit)

# Analyze
#--------
# Plot a histogram
plot_histogram(counts)
# END
```

Figura 17: Algorisme de Deutsch que hem enviat a l'ordinador quàntic

RESULTATS OBTINGUTS

Un cop l'execució ha finalitzat i el backend ens ha tornat el resultat encapsulat en l'objecte job, podem veure que l'estat d'aquest ha passat de: `job is queded (None)` a `job has successfully run`. A més ens ha retornat el comptatge del nombre de cops d'entre les 1000 execucions(shots) en què ha sortit cada estat.

Com podem veure a continuació:

```
Job Status: job has successfully run

Total count for 0 and 1 are: {'00': 7, '01': 26, '10': 45, '11': 922}
```

Figura 18: Resultats de l'execució de l'algorisme

On veiem que d'entre les 1000 execucions, un total de 922 han retornat l'estat 11 com a resultat. 7 execucions han retornat l'estat 00, 26 l'estat 01 i 45 l'estat 10.

Si som observadors ara ja no tenim que 1000 de les 1000 execucions ens retornen l'estat 11. I per tant que amb una probabilitat del 100% sempre ens surti l'estat 11.Si no, que ara també tenim execucions que retornen altres estats.

Aquesta variabilitat en els resultats és deguda a diverses fonts d'errors inherents a la tecnologia quàntica. Algunes d'aquestes fluctuacions en els resultats són degudes a:

- **Soroll quàntic:** És una font d'errors que es troba en els qubits i en les portes quàntiques del processador. A mesura que augmenta la complexitat del circuit i s'apliquen més operacions, l'acumulació de soroll ens afecta a la precisió dels resultats.
- **Errors en les portes quàntiques:** Les portes quàntiques no són perfectes i poden introduir errors durant l'execució del circuit. Aquests errors poden ser causats per desviacions en els paràmetres de les portes quàntiques.
  En el nostre cas, podem veure l'error de les portes quàntiques en l'ordinador quàntic de Manila on hem executat l'algorisme.

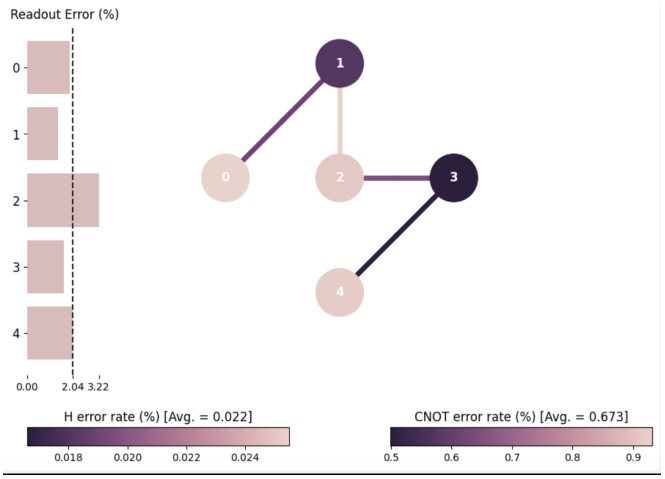


Figura 19: Errors en les portes quàntiques de l'ordinador quàntic sobre el que hem executat l'experiment

- **Calibratge:** Els processadors quàntics requereixen un calibratge i una configuració precisa per funcionar adequadament.
- **Efectes de l'entorn:** Els ordinadors quàntics són sensibles a les pertorbacions ambientals, com ara les fluctuacions tèrmiques o electromagnètiques. Aquests efectes de l'entorn poden provocar errors i variabilitat en els resultats.

Tot i aquestes variacions en els resultats, actualment s'implementen tècniques de correcció d'errors, es realitza una millora en els protocols de calibratge i

s'introdueixen mètodes per reduir el soroll i les interferències per produir resultats més precisos i fiables[3].
Aquí podem veure el circuit quàntic resultant que és el que hem simulat:

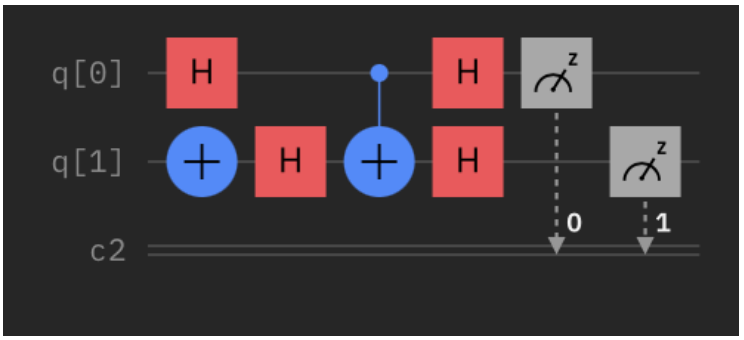


Figura 20: Circuit resultant de l'execució

També, tal i com havíem fet de forma local, podem plotejar el resultats del comptatge de cada estat en forma de histograma:

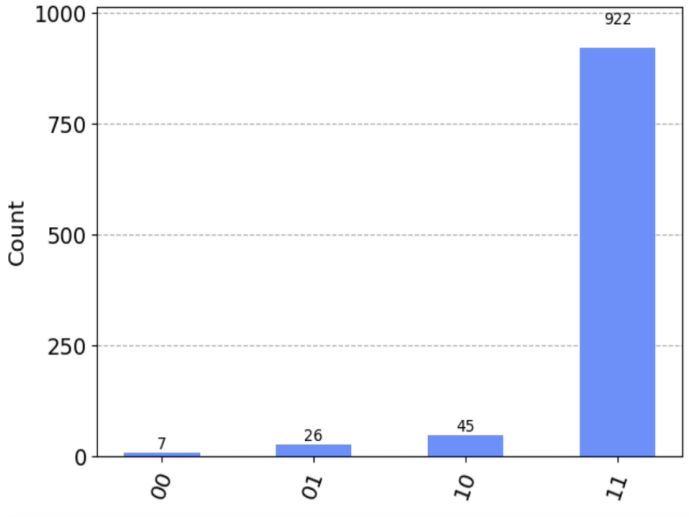


Figura 21: Histograma amb el resultat de l'execució a l'ordinador quàntic de Manila

Si tornem a executar l'algorisme i tornem a enviar el job a l'ordinador quàntic de Manila es pot veure que obtenim resultats diferents, tal i com esperàvem.

[3] Es pot consultar aquest paper per veure un estudi de la reliability dels ordinadors quantics de iBM: https://reunir.unir.net/bitstream/handle/123456789/14591/ip2023_04_005.pdf?sequence=1&isAllowed=y

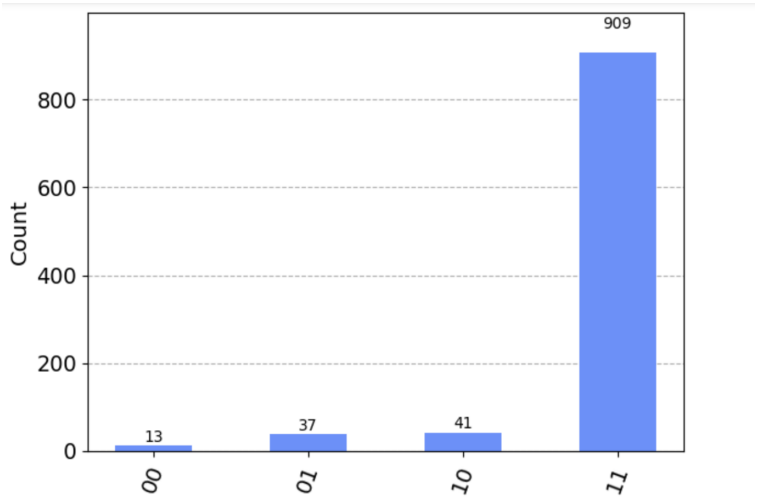


Figura 22: Histograma amb una altre execució al mateix ordinador quàntic

EXEMPLES GUIATS

L'informe que s'ha elaborat és una petita introducció de com podem construir circuits i fer els nostres algorismes quàntics amb l'entorn de Qiskit que ens proporciona IBM. Es poden consultar més exemples i algorismes a la pàgina: https://qiskit.org/learn/.