TrabalhoIC

June 6, 2021

1 Trabalho prático Interação e Concorrência 2020/2021

Ricardo Silva a71532

José Barbosa a69136

1.0.1 Resumo:

Neste trabalho temos de implementar um algoritmo em Quiskit para encontrar o número associado ao nosso grupo numa lista.O número do nosso grupo é o 11, teremos que encontrar o qubit |011>,pois 3 11 mod 8 e 3 = 2^1 + 2^0. Como a lista de numeros varia entre 0 e 7 serão nessessarios 3 qubits.

1.0.2 Problema:

Each group of students has a number assigned, N. Now, you have to use a quantum algorithm to find s

$$s = N mod 8$$

Implement the correct algorithm in a Jupyter Notebook file. Each work should contain (and will be evaluated on) the following steps:

1 - Division of the algorithm into sections; Utilisation of the state vector simulator to explain each step (special attention to the oracle);

Algoritmo de Grover

Trusted Notebook” width=”800 px” align=”center”>

-Inicialização;

-Oráculo;

-Amplificação.

Optamos por implementar o algoritmo de Grover. Consideremos então uma lista constituída por elementos de 0 a 7, isto é, 000 a 111. Pretendemos localizar um objeto em particular, o 011. Portanto, temos:

0	0	0	1	0	0	0	0
000	001	010	011	100	101	110	111

O algoritmo de Grover necessida de um Oráculo. Uma forma simples de codificar a função do Oráculo é:

$$() = 0$$
$$(w) = 0$$

1.0.3 Inicialização

[106]:
```
from qiskit import *
%matplotlib inline
from qiskit.tools.visualization import *
```

[107]:
```
barriers=barriers = True
#Step 0
nqubits=3

#Criar circuito
qc=QuantumCircuit(nqubits)
```

[108]:
```
#inicializar o sistema com a mesma amplitude
#em todos os estados de entrada possíveis.
#aplicar as portas de Hadammard aos qubits
#de forma a criar uma sobreposição uniforme

for q in range(nqubits):
    qc.h(q)

if barriers:
    qc.barrier()
```

1.0.4 Oraculo

O Oráculo identifica a solução para o problema, ou seja, o qubit $|011\rangle$. Então, a fase do estado $|011\rangle$ faz uma rotação de radianos. Essa transformação significa que a amplitude do estado $|011\rangle$ ficou negativa o que significa que a amplitude média foi reduzida.Para definir o oráculo, para além da Haddammard gate, recorremos a Pauli-X gate e a Controlled-X gate.

[109]:
```
#Step 2: Marcar estado |011>  usando o oraculo:
qc.h(0)
qc.x(2)
qc.ccx(2,1,0)
```

```
qc.h(0)
qc.x(2)

if barriers:
    qc.barrier()
```

1.0.5 Amplificação

Nesta fase aumenta a amplitude do qubit e volta a inverte-lo.

[110]:
```
#Step 3:  Realizar a reflexão em torno da amplitude média

#Aplicar Hadamard e X gates aos qubits

for q in range(nqubits):
    qc.h(q)
    qc.x(q)

#Aplicar a doubly controlled Z gate entre qubits
qc.h(0)
qc.ccx(2,1,0)
qc.h(0)
# Aplicar X gates e Hadamard gates aos qubits
for q in range(nqubits):
    qc.x(q)
    qc.h(q)
```

[111]:
```
import math as m
times=m.sqrt(2**(nqubits))
print(times)
```

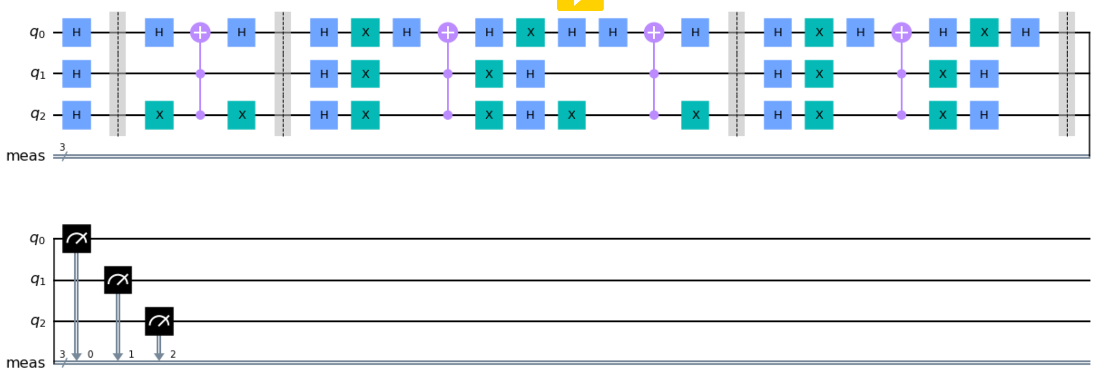
```
2.8284271247461903
```

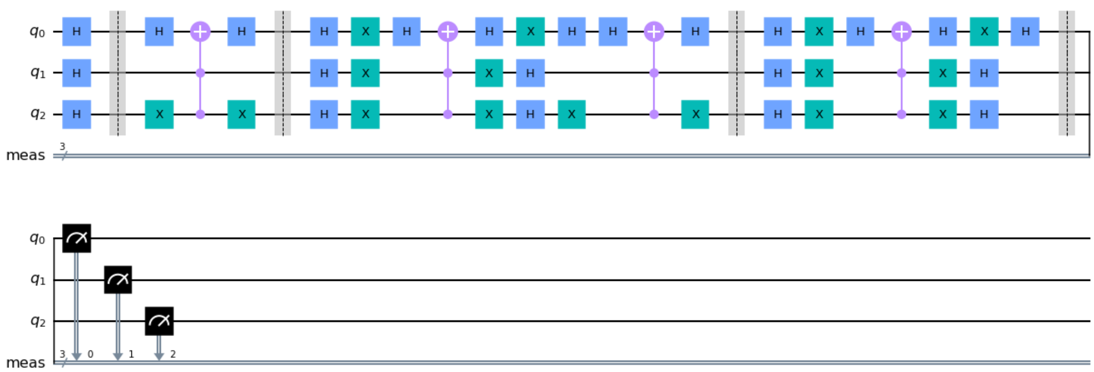
[112]:
```
#Step 4: Repetir step 3 e step 2 de modo a aproximar a medida
#ideal e obter bons resultados.
qc.h(0)
qc.x(2)
qc.ccx(2,1,0)
qc.h(0)
qc.x(2)
if barriers:
    qc.barrier()

for q in range(nqubits):
    qc.h(q)
    qc.x(q)
```

```
qc.h(0)
qc.ccx(2,1,0)
qc.h(0)

for q in range(nqubits):
    qc.x(q)
    qc.h(q)
```

[113]:
```
#Step 5: Medir os 3 qubits para receber estados |011>
qc.measure_all()
qc.draw(output='mpl')
```

[113]:



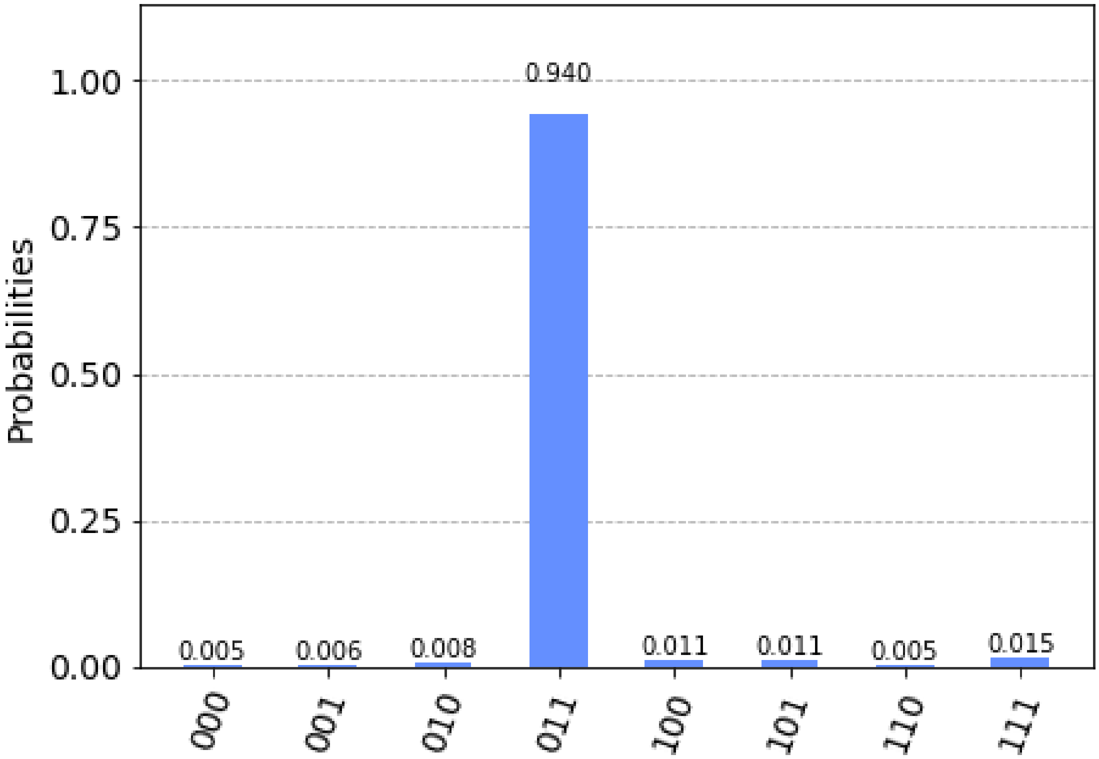


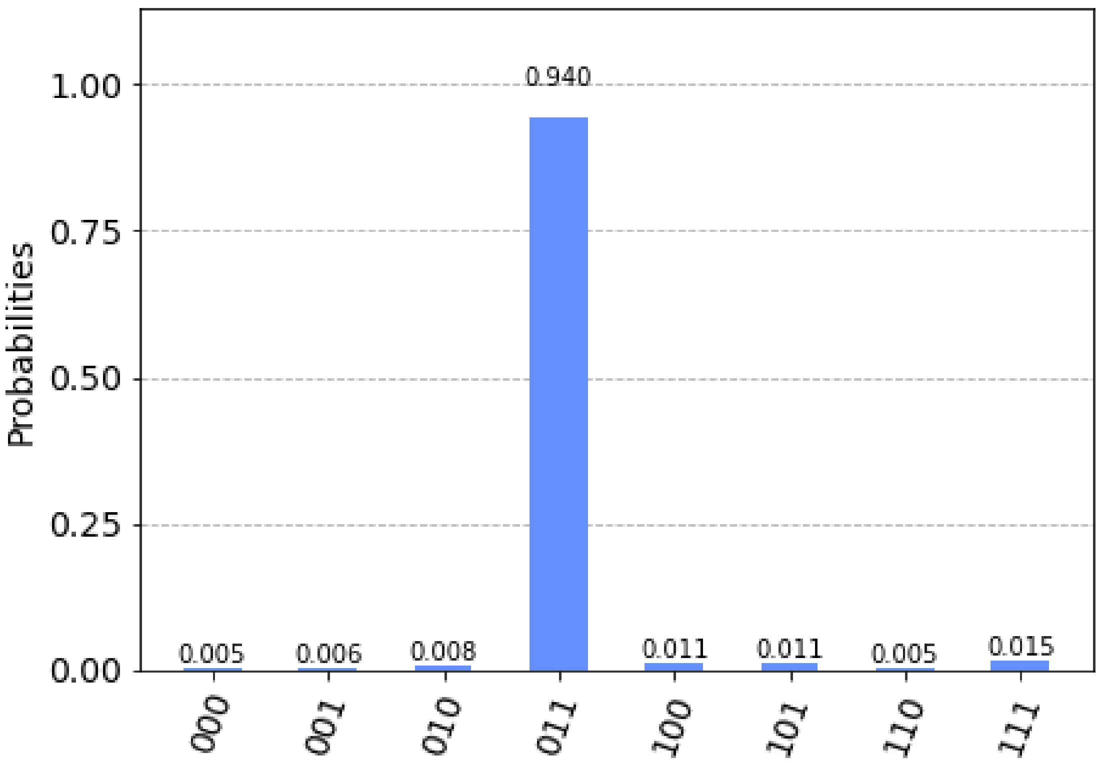
2 - Application of noise simulator to predict the best optimisation;

[114]:
```
backend=Aer.get_backend('qasm_simulator')
shots=1024
#Executa a lista de circuitos quânticos no backend e guarda o resultado
```

```
results=execute(qc, backend=backend, shots=shots).result()
#Obter os dados para o histograma
answer=results.get_counts(qc)
#Desenhar o histograma
plot_histogram(answer)
```

[114]:





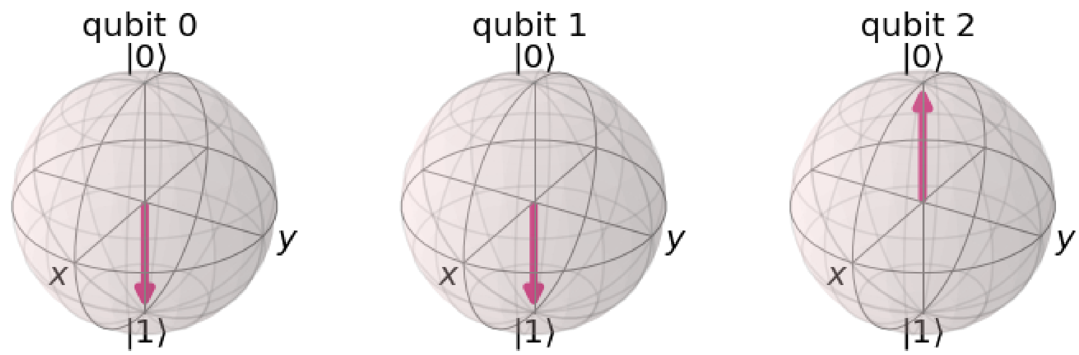
[115]:
```
#calcula a profundidade do circuito
qc.depth()
```

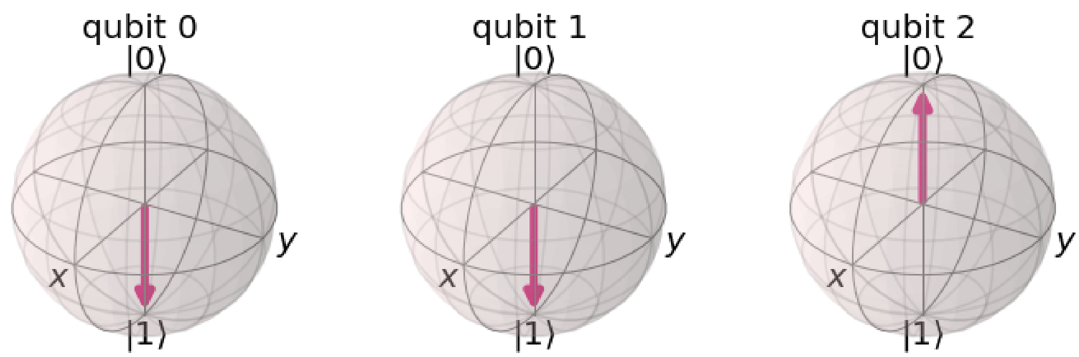
[115]: 22

[116]:
```
backend_state = Aer.get_backend('statevector_simulator')
#Executa a lista de circuitos quânticos no backend e guarda o resultado
result = execute(qc, backend_state).result() #Obtém o statevector final de um␣
↪circuito quântico
psi2 = result.get_statevector(qc)
#Desenho das esferas de Bloch e as respetivas projeções em cada eixo
plot_bloch_multivector(psi2)
```

```
/opt/conda/lib/python3.8/site-packages/qiskit/visualization/bloch.py:69:
MatplotlibDeprecationWarning:
The M attribute was deprecated in Matplotlib 3.4 and will be removed two minor
releases later. Use self.axes.M instead.
  x_s, y_s, _ = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M)
```

[116]:



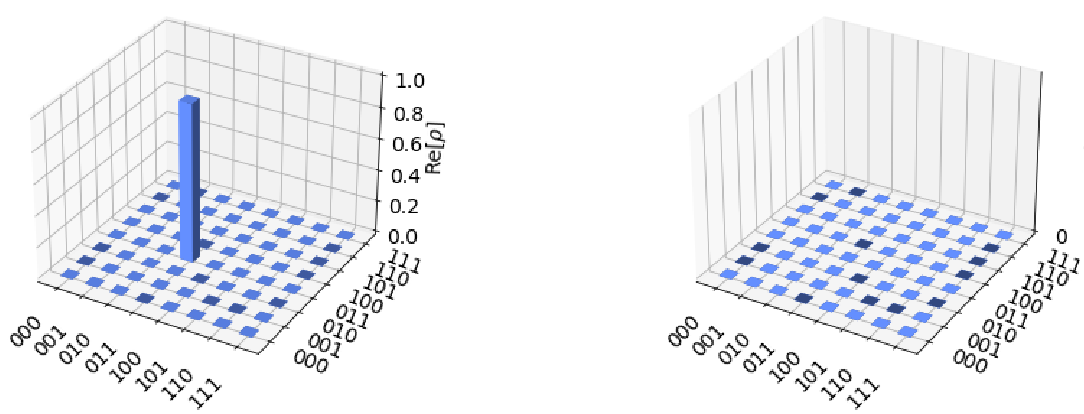


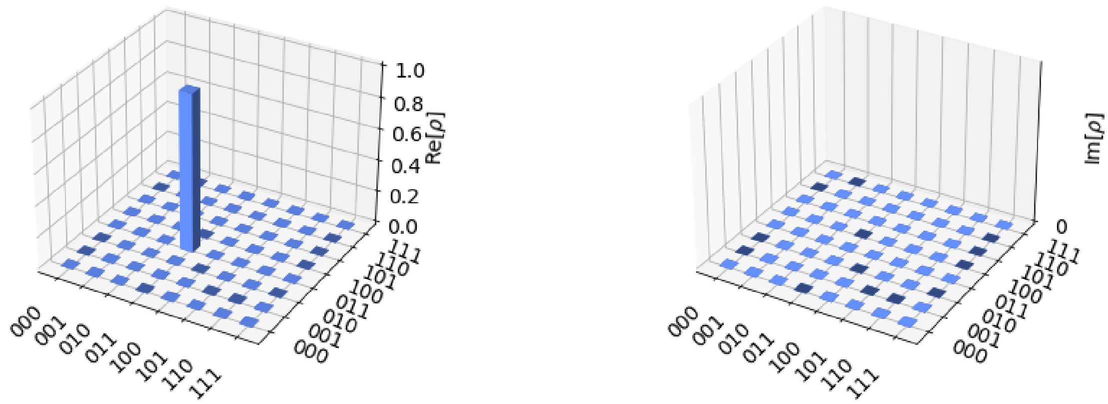
[117]:
```
psi2.real
```

[117]:
```
array([ 0.,  0.,  0.,  1.,  0.,  0.,  0., -0.])
```

[118]:
```
#Traça dois gráficos com barras 3D (bidimensionais) da parte real # e parte imaginária da matriz de densidade
plot_state_city(psi2)
```

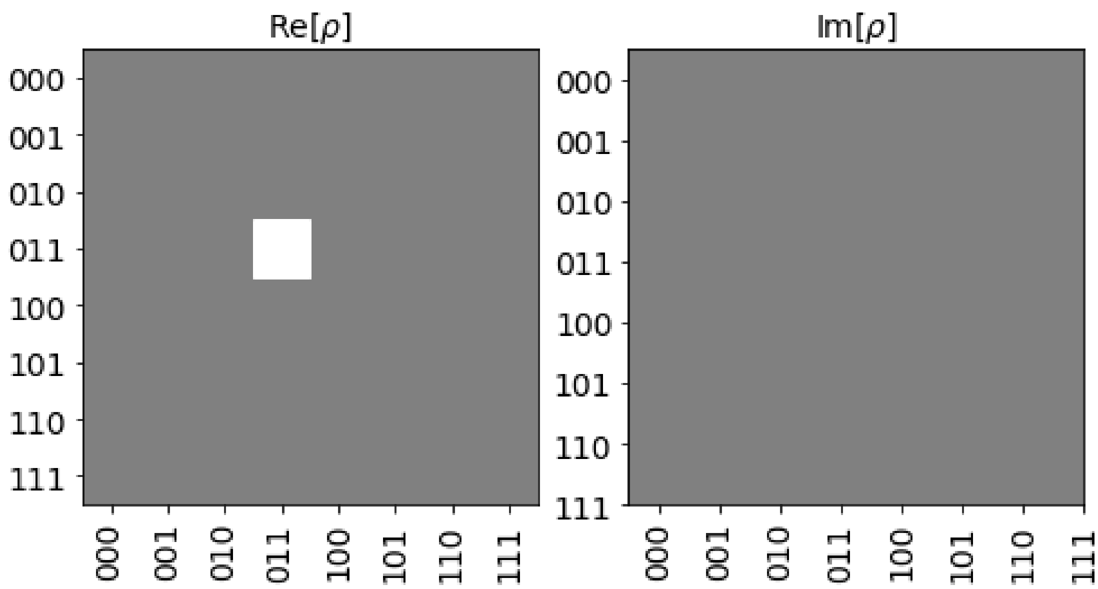
[118]:

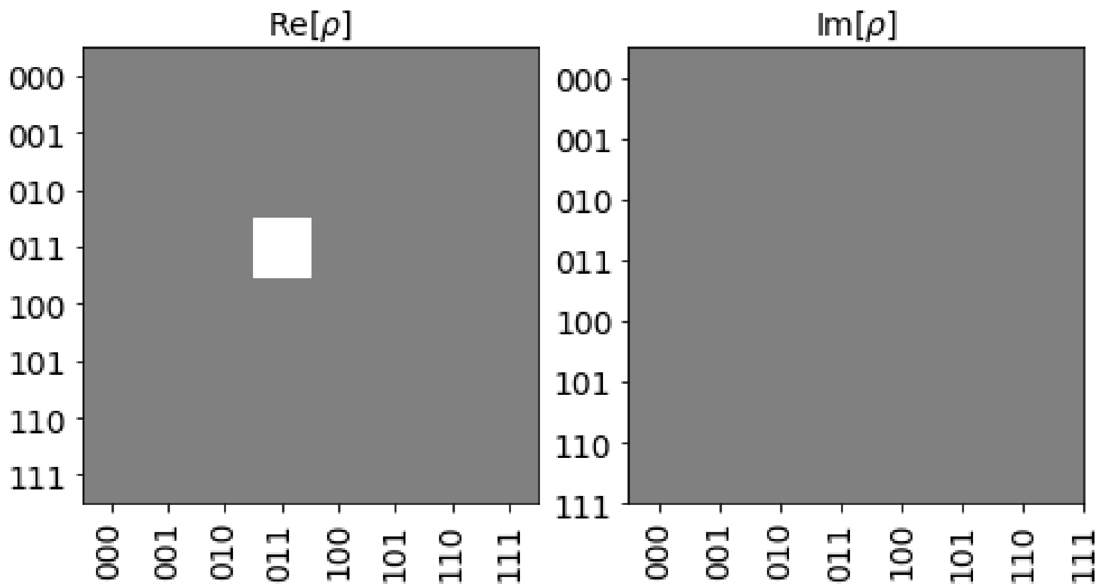




[119]:
```
#Faz um diagrama de hinton para o estado quântico
plot_state_hinton(psi2)
```

[119]:





3 - Execution in an IBM Q backend.

[120]:
```
from qiskit import IBMQ
from qiskit.providers.ibmq import least_busy

# Carrega as informações da conta local e as respectivas máquinas associadas.
provider = IBMQ.load_account()
provider.backends()
```

```
ibmqfactory.load_account:WARNING:2021-06-05 23:13:48,562: Credentials are
already in use. The existing account in the session will be replaced.
```

[120]:
```
[<IBMQSimulator('ibmq_qasm_simulator') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmqx2') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_16_melbourne') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_armonk') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_athens') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_santiago') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_lima') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_belem') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_quito') from IBMQ(hub='ibm-q', group='open',
```

```
project='main')>,
 <IBMQSimulator('simulator_statevector') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQSimulator('simulator_mps') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQSimulator('simulator_extended_stabilizer') from IBMQ(hub='ibm-q',
group='open', project='main')>,
 <IBMQSimulator('simulator_stabilizer') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_manila') from IBMQ(hub='ibm-q', group='open',
project='main')>]
```

[121]:
```
# Backend overview
import qiskit.tools.jupyter
%qiskit_backend_overview
```

```
VBox(children=(HTML(value="<h2 style ='color:#ffffff; background-color:#000000;padding-top: 1%
```

[122]:
```
from qiskit.tools.monitor import backend_overview, backend_monitor
#Fornece informações gerais de todos os backends do IBMQ que estão disponíveis
backend_overview()
```

```
ibmq_manila                 ibmq_quito                  ibmq_belem
-----------                 ----------                  ----------
Num. Qubits:  5             Num. Qubits:  5             Num. Qubits:  5
Pending Jobs: 14            Pending Jobs: 7             Pending Jobs: 1
Least busy:   False         Least busy:   False         Least busy:   False
Operational:  True          Operational:  True          Operational:  True
Avg. T1:      145.8         Avg. T1:      75.2          Avg. T1:      79.3
Avg. T2:      67.0          Avg. T2:      73.2          Avg. T2:      91.6



ibmq_lima                   ibmq_santiago               ibmq_athens
---------                   -------------               -----------
Num. Qubits:  5             Num. Qubits:  5             Num. Qubits:  5
Pending Jobs: 10            Pending Jobs: 7             Pending Jobs: 0
Least busy:   False         Least busy:   False         Least busy:   True
Operational:  True          Operational:  True          Operational:  True
Avg. T1:      69.2          Avg. T1:      141.6         Avg. T1:      96.1
Avg. T2:      64.9          Avg. T2:      136.4         Avg. T2:      120.6



ibmq_armonk                 ibmq_16_melbourne           ibmqx2
-----------                 -----------------           ------
Num. Qubits:  1             Num. Qubits:  15            Num. Qubits:  5
```

```
Pending Jobs: 0          Pending Jobs: 1          Pending Jobs: 1
Least busy:   False      Least busy:   False      Least busy:   False
Operational:  True       Operational:  True       Operational:  True
Avg. T1:      124.6      Avg. T1:      57.5       Avg. T1:      54.1
Avg. T2:      217.3      Avg. T2:      56.2       Avg. T2:      40.5
```

[123]:

```python
#Seleciona uma máquina
backend_device = provider.get_backend('ibmq_quito')
print("Running on: ", backend_device)
```

```
Running on:  ibmq_quito
```

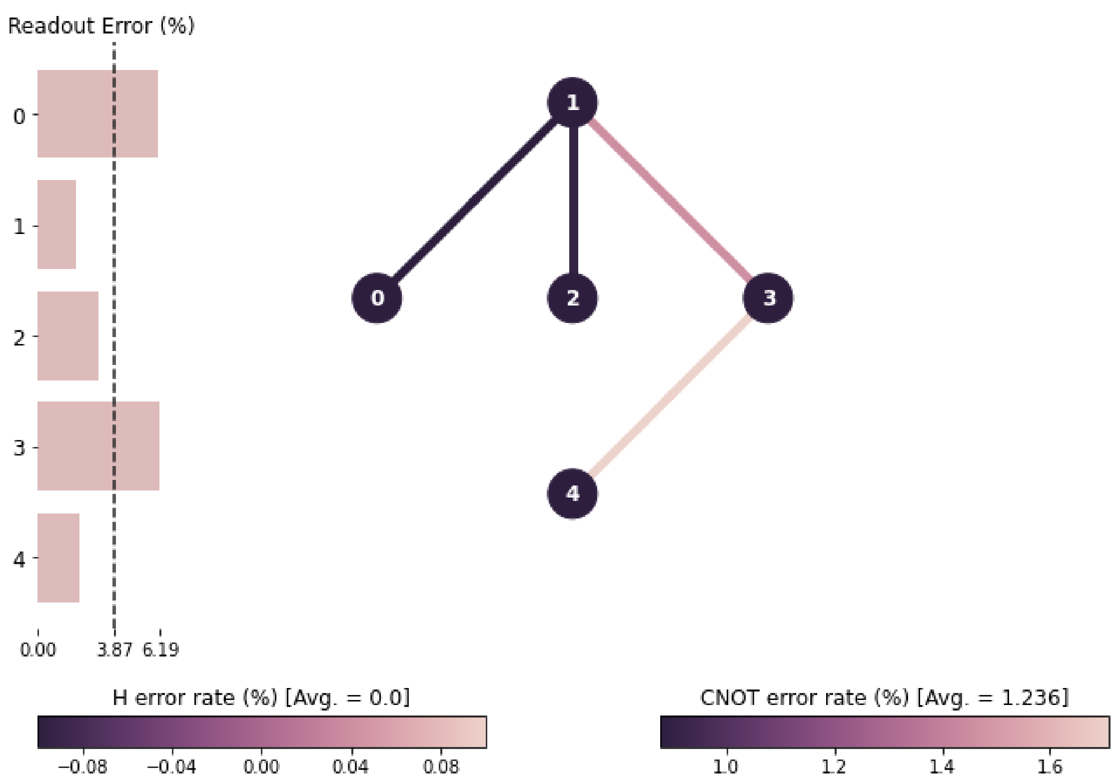
[124]:

```python
#Obtém informação sobre o backend

backend_device
```

```
VBox(children=(HTML(value="<h1 style='color:#ffffff;background-color:#000000;padding-top: 1%;pa
```

[124]:

```
<IBMQBackend('ibmq_quito') from IBMQ(hub='ibm-q', group='open', project='main')>
```



[125]:
```
backend_monitor(backend_device)
```

```
ibmq_quito
==========
Configuration
-------------
    n_qubits: 5
    operational: True
    status_msg: active
    pending_jobs: 6
    backend_version: 1.1.2
    basis_gates: ['id', 'rz', 'sx', 'x', 'cx', 'reset']
    local: False
    simulator: False
    default_rep_delay: 250.0
    open_pulse: False
    qubit_channel_mapping: [['u0', 'm0', 'u1', 'd0'], ['u3', 'u4', 'u1', 'u0',
'm1', 'u2', 'u5', 'd1'], ['d2', 'm2', 'u4', 'u2'], ['u3', 'm3', 'd3', 'u5',
'u6', 'u7'], ['d4', 'm4', 'u6', 'u7']]
    description: 5 qubit device Quito
    pulse_num_channels: 9
    coupling_map: [[0, 1], [1, 0], [1, 2], [1, 3], [2, 1], [3, 1], [3, 4], [4,
3]]
    dynamic_reprate_enabled: True
    backend_name: ibmq_quito
    allow_q_object: True
    meas_levels: [1, 2]
    supported_instructions: ['u3', 'u1', 'setf', 'acquire', 'delay', 'x',
'play', 'measure', 'rz', 'sx', 'id', 'u2', 'shiftf', 'reset', 'cx']
    max_experiments: 75
    credits_required: True
    n_uchannels: 8
    hamiltonian: {'description': 'Qubits are modeled as Duffing oscillators. In
this case, the system includes higher energy states, i.e. not just |0> and |1>.
The Pauli operators are generalized via the following set of
transformations:\n\n$(\\mathbb{I}-\\sigma_{i}^z)/2 \\rightarrow O_i \\equiv
b^\\dagger_{i} b_{i}$,\n\n$\\sigma_{+} \\rightarrow b^\\dagger$,\n\n$\\sigma_{-}
\\rightarrow b$,\n\n$\\sigma_{i}^X \\rightarrow b^\\dagger_{i} +
b_{i}$.\n\nQubits are coupled through resonator buses. The provided Hamiltonian
has been projected into the zero excitation subspace of the resonator buses
leading to an effective qubit-qubit flip-flop interaction. The qubit resonance
frequencies in the Hamiltonian are the cavity dressed frequencies and not
exactly what is returned by the backend defaults, which also includes the
dressing due to the qubit-qubit interactions.\n\nQuantities are returned in
angular frequencies, with units 2*pi*GHz.\n\nWARNING: Currently not all system
```

```
Hamiltonian information is available to the public, missing values have been
replaced with 0.\n', 'h_latex': '\\begin{align} \\mathcal{H}/\\hbar = & \\sum_{i
=0}^{4}\\left(\\frac{\\omega_{q,i}}{2}(\\mathbb{I}-\\sigma_i^{z})+\\frac{\\Delta
_{i}}{2}(O_i^2-O_i)+\\Omega_{d,i}D_i(t)\\sigma_i^{X}\\right) \\\\ & +
J_{0,1}(\\sigma_{0}^{+}\\sigma_{1}^{-}+\\sigma_{0}^{-}\\sigma_{1}^{+}) +
J_{1,3}(\\sigma_{1}^{+}\\sigma_{3}^{-}+\\sigma_{1}^{-}\\sigma_{3}^{+}) +
J_{3,4}(\\sigma_{3}^{+}\\sigma_{4}^{-}+\\sigma_{3}^{-}\\sigma_{4}^{+}) +
J_{1,2}(\\sigma_{1}^{+}\\sigma_{2}^{-}+\\sigma_{1}^{-}\\sigma_{2}^{+}) \\\\ & +
\\Omega_{d,0}(U_{0}^{(0,1)}(t))\\sigma_{0}^{X} +
\\Omega_{d,1}(U_{1}^{(1,0)}(t)+U_{3}^{(1,3)}(t)+U_{2}^{(1,2)}(t))\\sigma_{1}^{X}
\\\\ & + \\Omega_{d,2}(U_{4}^{(2,1)}(t))\\sigma_{2}^{X} +
\\Omega_{d,3}(U_{5}^{(3,1)}(t)+U_{6}^{(3,4)}(t))\\sigma_{3}^{X} \\\\ & +
\\Omega_{d,4}(U_{7}^{(4,3)}(t))\\sigma_{4}^{X} \\\\ \\end{align}', 'h_str':
['_SUM[i,0,4,wq{i}/2*(I{i}-Z{i})]', '_SUM[i,0,4,delta{i}/2*O{i}*O{i}]',
'_SUM[i,0,4,-delta{i}/2*O{i}]', '_SUM[i,0,4,omegad{i}*X{i}||D{i}]',
'jq0q1*Sp0*Sm1', 'jq0q1*Sm0*Sp1', 'jq1q3*Sp1*Sm3', 'jq1q3*Sm1*Sp3',
'jq3q4*Sp3*Sm4', 'jq3q4*Sm3*Sp4', 'jq1q2*Sp1*Sm2', 'jq1q2*Sm1*Sp2',
'omegad1*X0||U0', 'omegad0*X1||U1', 'omegad3*X1||U3', 'omegad2*X1||U2',
'omegad1*X2||U4', 'omegad1*X3||U5', 'omegad4*X3||U6', 'omegad3*X4||U7'], 'osc':
{}, 'qub': {'0': 3, '1': 3, '2': 3, '3': 3, '4': 3}, 'vars': {'delta0':
-2.0827513300148186, 'delta1': -2.0058778622715616, 'delta2':
-2.0880065449040663, 'delta3': -2.1053738129998156, 'delta4':
-2.005987719908062, 'jq0q1': 0.011708244917214646, 'jq1q2':
0.012036525934936945, 'jq1q3': 0.011454603594507371, 'jq3q4':
0.010153599217904036, 'omegad0': 1.1235895123483226, 'omegad1':
1.3360010853272766, 'omegad2': 1.1089676175614007, 'omegad3':
1.3638178453389296, 'omegad4': 0.5491008728368838, 'wq0': 33.30330436045644,
'wq1': 31.923922590354472, 'wq2': 33.44083805070931, 'wq3': 32.44511909152861,
'wq4': 31.746360666232587}}
    max_shots: 8192
    n_registers: 1
    rep_times: [0.001]
    rep_delay_range: [0.0, 500.0]
    parametric_pulses: ['gaussian', 'gaussian_square', 'drag', 'constant']
    pulse_num_qubits: 3
    meas_lo_range: [[6.807868157e+18, 7.807868157e+18], [6.729269527e+18,
7.729269527e+18], [6.991060108e+18, 7.991060108e+18], [6.854751046e+18,
7.854751046e+18], [6.904365522e+18, 7.904365522e+18]]
    conditional: False
    acquisition_latency: []
    multi_meas_enabled: True
    url: None
    meas_map: [[0, 1, 2, 3, 4]]
    conditional_latency: []
    processor_type: {'family': 'Falcon', 'revision': 4, 'segment': 'T'}
    allow_object_storage: True
    discriminators: ['linear_discriminator', 'quadratic_discriminator',
'hw_centroid']
```

quantum_volume: 16
uchannels_enabled: True
input_allowed: ['job']
u_channel_lo: [[{'q': 1, 'scale': (1+0j)}], [{'q': 0, 'scale': (1+0j)}], [{'q': 2, 'scale': (1+0j)}], [{'q': 3, 'scale': (1+0j)}], [{'q': 1, 'scale': (1+0j)}], [{'q': 1, 'scale': (1+0j)}], [{'q': 4, 'scale': (1+0j)}], [{'q': 3, 'scale': (1+0j)}]]
memory: True
sample_name: family: Falcon, revision: 4, segment: T
meas_kernels: ['hw_boxcar']
dt: 0.2222222222222222
qubit_lo_range: [[4.800385510260514e+18, 5.800385510260514e+18], [4.5808500831379374e+18, 5.580850083137937e+18], [4.822274676905928e+18, 5.822274676905928e+18], [4.663801082622002e+18, 5.663801082622002e+18], [4.552590225209032e+18, 5.552590225209032e+18]]
channels: {'acquire0': {'operates': {'qubits': [0]}, 'purpose': 'acquire', 'type': 'acquire'}, 'acquire1': {'operates': {'qubits': [1]}, 'purpose': 'acquire', 'type': 'acquire'}, 'acquire2': {'operates': {'qubits': [2]}, 'purpose': 'acquire', 'type': 'acquire'}, 'acquire3': {'operates': {'qubits': [3]}, 'purpose': 'acquire', 'type': 'acquire'}, 'acquire4': {'operates': {'qubits': [4]}, 'purpose': 'acquire', 'type': 'acquire'}, 'd0': {'operates': {'qubits': [0]}, 'purpose': 'drive', 'type': 'drive'}, 'd1': {'operates': {'qubits': [1]}, 'purpose': 'drive', 'type': 'drive'}, 'd2': {'operates': {'qubits': [2]}, 'purpose': 'drive', 'type': 'drive'}, 'd3': {'operates': {'qubits': [3]}, 'purpose': 'drive', 'type': 'drive'}, 'd4': {'operates': {'qubits': [4]}, 'purpose': 'drive', 'type': 'drive'}, 'm0': {'operates': {'qubits': [0]}, 'purpose': 'measure', 'type': 'measure'}, 'm1': {'operates': {'qubits': [1]}, 'purpose': 'measure', 'type': 'measure'}, 'm2': {'operates': {'qubits': [2]}, 'purpose': 'measure', 'type': 'measure'}, 'm3': {'operates': {'qubits': [3]}, 'purpose': 'measure', 'type': 'measure'}, 'm4': {'operates': {'qubits': [4]}, 'purpose': 'measure', 'type': 'measure'}, 'u0': {'operates': {'qubits': [0, 1]}, 'purpose': 'cross-resonance', 'type': 'control'}, 'u1': {'operates': {'qubits': [1, 0]}, 'purpose': 'cross-resonance', 'type': 'control'}, 'u2': {'operates': {'qubits': [1, 2]}, 'purpose': 'cross-resonance', 'type': 'control'}, 'u3': {'operates': {'qubits': [1, 3]}, 'purpose': 'cross-resonance', 'type': 'control'}, 'u4': {'operates': {'qubits': [2, 1]}, 'purpose': 'cross-resonance', 'type': 'control'}, 'u5': {'operates': {'qubits': [3, 1]}, 'purpose': 'cross-resonance', 'type': 'control'}, 'u6': {'operates': {'qubits': [3, 4]}, 'purpose': 'cross-resonance', 'type': 'control'}, 'u7': {'operates': {'qubits': [4, 3]}, 'purpose': 'cross-resonance', 'type': 'control'}}
dtm: 0.2222222222222222
online_date: 2021-01-08 05:00:00+00:00

Qubits [Name / Freq / T1 / T2 / RZ err / SX err / X err / Readout err]

Q0 / 5.30039 GHz / 46.36425 us / 41.15058 us / 0.00000 / 0.00033 / 0.00033 / 0.06030

```
    Q1 / 5.08085 GHz / 78.98568 us / 93.24751 us / 0.00000 / 0.00026 / 0.00026 /
0.01930
    Q2 / 5.32227 GHz / 90.39715 us / 74.45609 us / 0.00000 / 0.00082 / 0.00082 /
0.03050
    Q3 / 5.16380 GHz / 28.67316 us / 13.04563 us / 0.00000 / 0.00097 / 0.00097 /
0.06190
    Q4 / 5.05259 GHz / 131.74298 us / 144.23578 us / 0.00000 / 0.00096 / 0.00096
/ 0.02150

Multi-Qubit Gates [Name / Type / Gate Error]
--------------------------------------------
    cx3_4 / cx / 0.01708
    cx4_3 / cx / 0.01708
    cx1_3 / cx / 0.01462
    cx3_1 / cx / 0.01462
    cx2_1 / cx / 0.00901
    cx1_2 / cx / 0.00901
    cx0_1 / cx / 0.00875
    cx1_0 / cx / 0.00875
```

[126]:

```
%qiskit_job_watcher
```

```
Accordion(children=(VBox(layout=Layout(max_width='710px', min_width='710px')),), layout=Layout

<IPython.core.display.Javascript object>
```

[127]:

```
#Executa a lista de circuitos quânticos no backend e guarda o resultado
job_r = execute(qc, backend_device, shots=shots)
jobID_r = job_r.job_id()
print('JOB ID: {}'.format(jobID_r))
```

```
JOB ID: 60bc067125cc6e3ffe65cb06
```

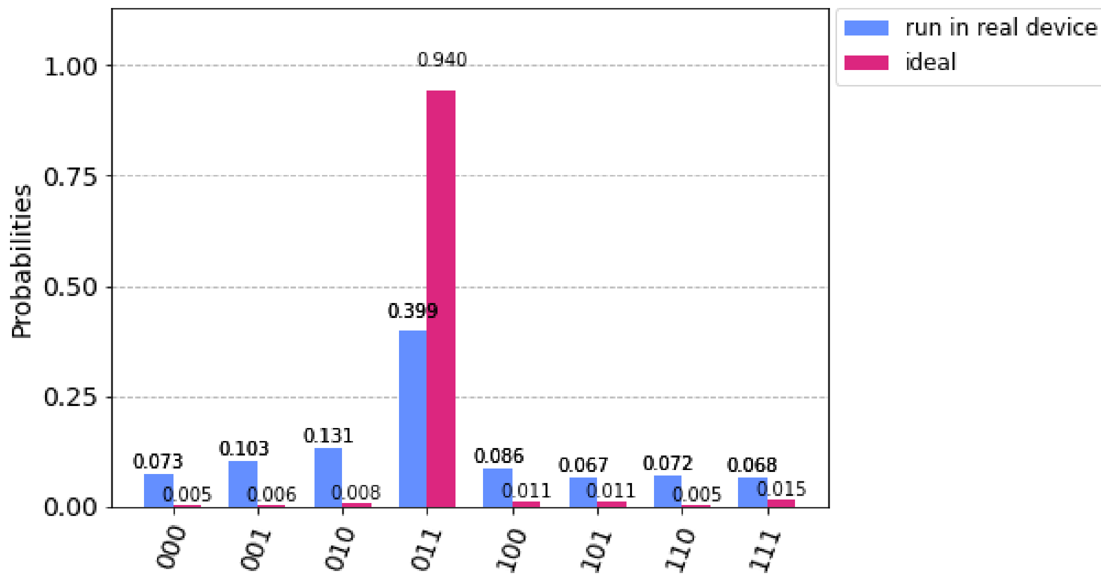
[128]:

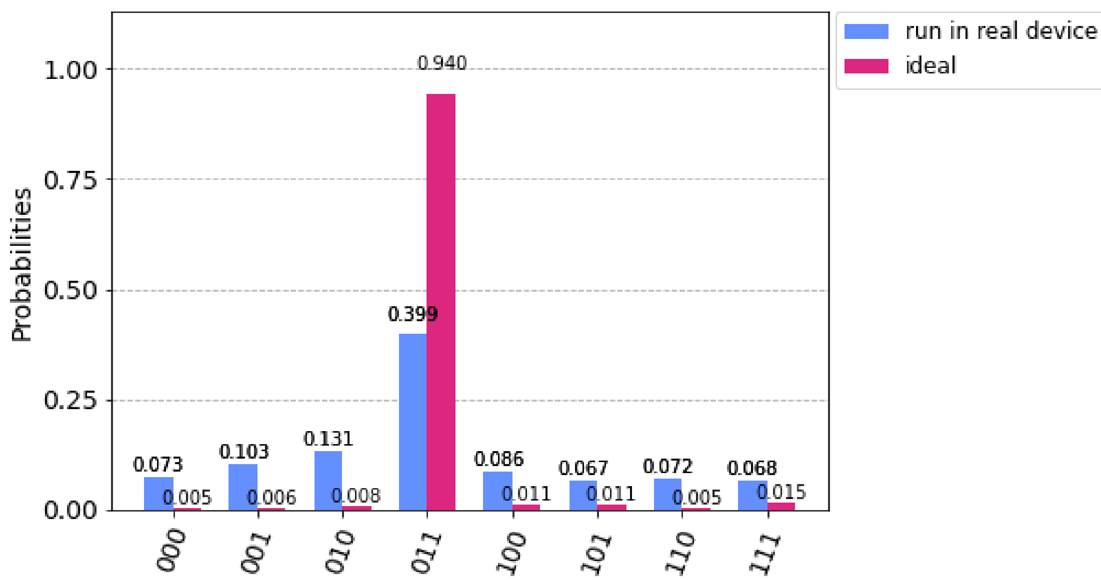
```
#ibmq_quito 1 times the oracle:
job_get=backend_device.retrieve_job("60bc067125cc6e3ffe65cb06")
result_r = job_get.result()
counts_run = result_r.get_counts(qc)
```

[129]:

```
 plot_histogram([counts_run, answer ], legend=[ 'run in real device', 'ideal'])
```

[129]:





4 - Mitigation of Error with Ignis.

Neste capitulo vamos tratar os erros com recurso ao módulo Ignis.

Sendo a computação quantica constituida por estados frageis, devido à sobreposição de qubits, este tipo de programação implica a possibilidade da existencia de ruido. O ruído pode aparecer aleatoriamente e fazer com que um qubit decaia do estado |1> para |0>.

[130]:
```
# Importe das funções de calibração
from qiskit.ignis.mitigation.measurement import (complete_meas_cal,␣
↪tensored_meas_cal,
                                                  CompleteMeasFitter,␣
↪TensoredMeasFitter)
```

Gerar uma lista de circuitos de calibração de medição. Cada circuito cria um estado básico. Uma vez que medimos 3 qubits, precisamos de 2^3 = 8 circuitos de calibração.

[131]:
```
# Gerar os circuitos de calibração
qr = QuantumRegister(nqubits)

# meas_calibs:
# list of quantum circuit objects containing the calibration circuits
# state_labels:
# calibration state labels
meas_calibs, state_labels = complete_meas_cal(qubit_list=[0,1,2], qr=qr,␣
↪circlabel='mcal')
```

[132]:
```
state_labels
```

[132]: ['000', '001', '010', '011', '100', '101', '110', '111']

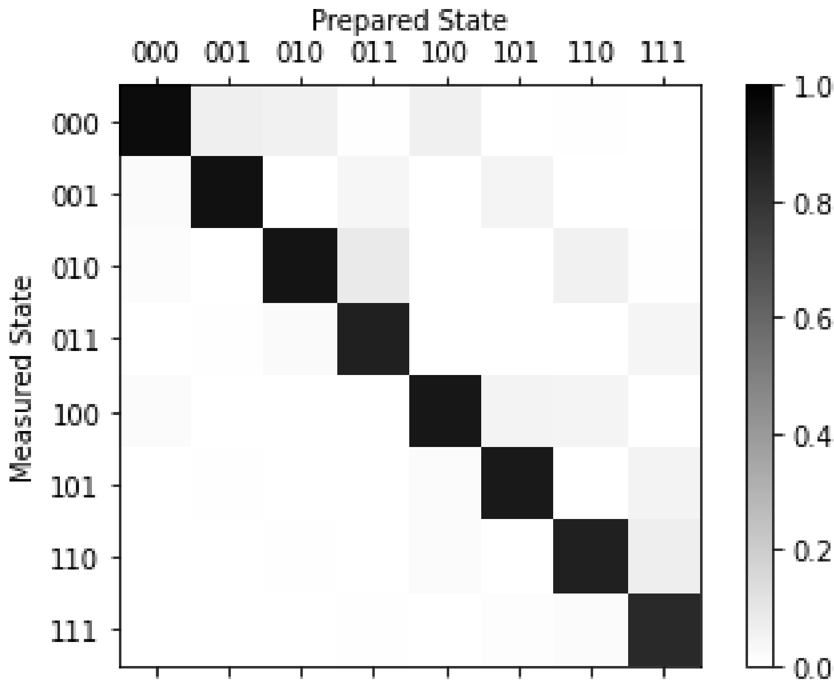
Calcular a matriz de calibração

[133]:
```
job_ignis = execute(meas_calibs, backend=backend_device, shots=shots)

jobID_run_ignis = job_ignis.job_id()

print('JOB ID: {}'.format(jobID_run_ignis))
```

JOB ID: 60bc0bf500aded86776a7462

[134]:
```
job_get=backend_device.retrieve_job("60bc0bf500aded86776a7462")

cal_results = job_get.result()
```

[135]:
```
meas_fitter = CompleteMeasFitter(cal_results, state_labels, circlabel='mcal')

# Plot the calibration matrix
meas_fitter.plot_calibration()
```



[136]:
```
# What is the measurement fidelity?
print("Average Measurement Fidelity: %f" % meas_fitter.readout_fidelity())
```

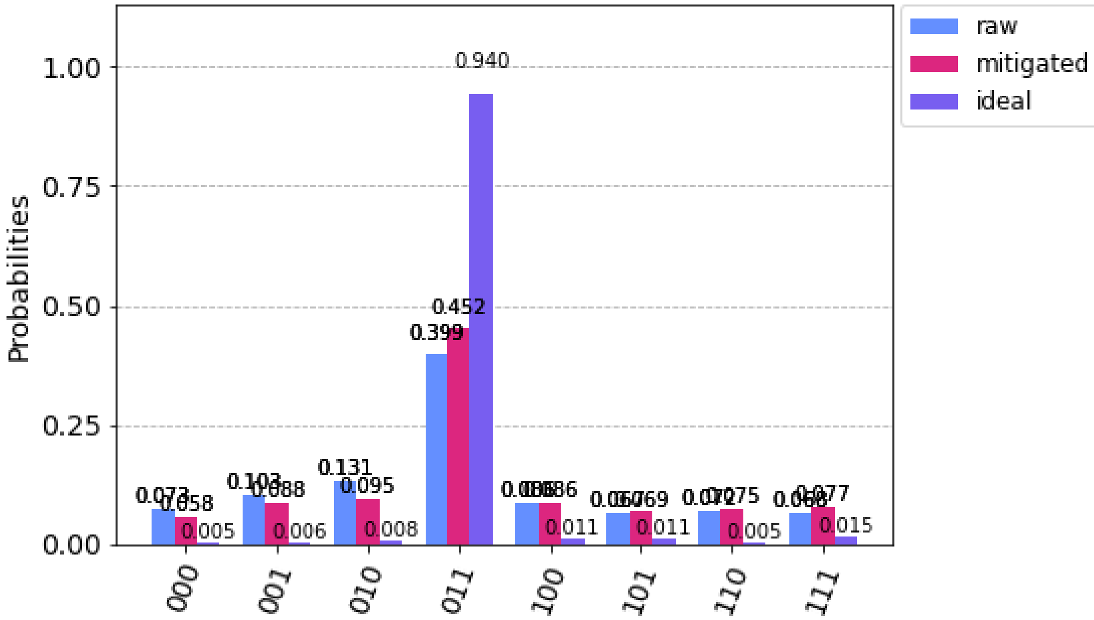
```
Average Measurement Fidelity: 0.896851
```

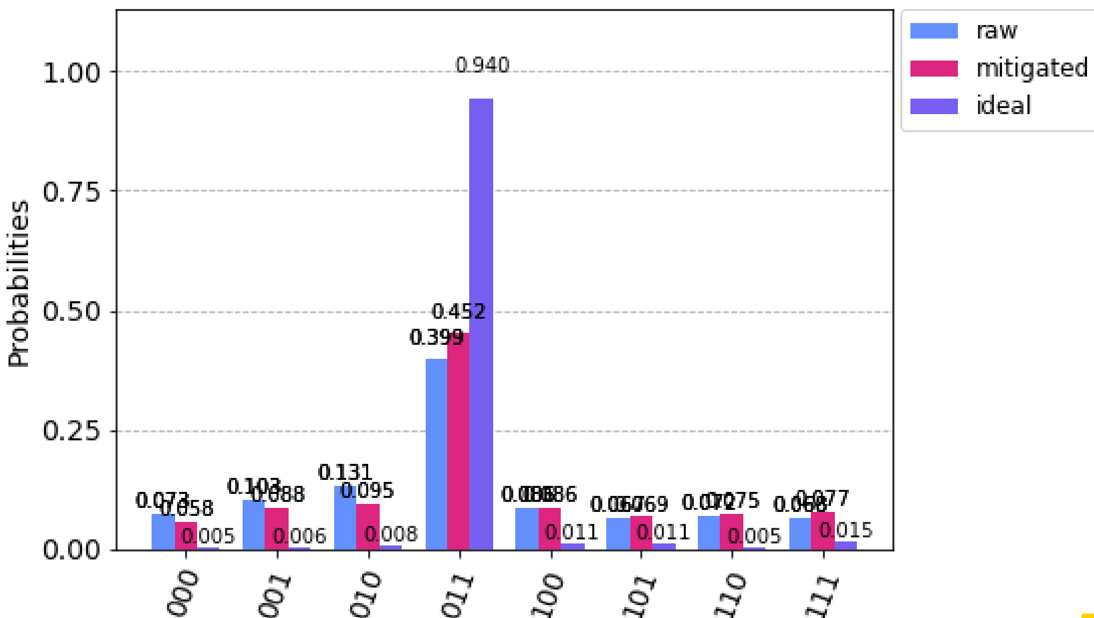
Aplicar a Calibração

[137]:
```
# Get the filter object
meas_filter = meas_fitter.filter

# Results with mitigation
mitigated_results = meas_filter.apply(result_r)
mitigated_counts = mitigated_results.get_counts()
```

[138]:
```
plot_histogram([counts_run, mitigated_counts, answer], legend=['raw',␣
↪'mitigated', 'ideal'])
```

[138]:


raw
mitigated
ideal
Probabilities
1.00
0.75
0.50
0.25
0.00
0.940
0.452
0.399
0.131
0.095
0.005
0.006
0.008
0.011
0.011
0.005
0.015
000
001
010
011
100
101
110
111


raw
mitigated
ideal
Probabilities
1.00
0.75
0.50
0.25
0.00
0.940
0.452
0.399
0.131
0.095
0.005
0.006
0.008
0.011
0.011
0.005
0.015
000
001
010
011
100
101
110
111