Trabalho_Pratico_IC

June 6, 2021

1 Trabalho Prático de IeC

Objetivo : Procurar um numero N em uma lista não ordenada, utilizando um algoritmo quantico. Dos algoritmos dados na aula, o algoritmo de groover é o mais adequado para efetuar a procura.

A implementação do algoritmo de Grover é feito em 3 fases: * Inicialização * Oraculo * Amplificação

`Preparação Quiskit:`

[1]:
```
from qiskit import Aer, IBMQ
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import execute, transpile

from qiskit.tools.visualization import plot_histogram, plot_state_city,␣
 ↪plot_state_hinton

import matplotlib.pyplot as plt
%matplotlib inline
```

Inicialização : Determinar o numero a ser procurado, e a sua representação binária:

[2]:
```
N = 14
w = N % 8
wb = bin(w)[2:]
print(w)
print(wb)
```

```
6
110
```

Selecionamos o número de qubits necessários para representar o número binário que nos foi dado. Neste caso são precisos 3 qbits.

[3]:
```
x=3
```

Oraculo : Criação do oraculo.

Decomposição CnZ

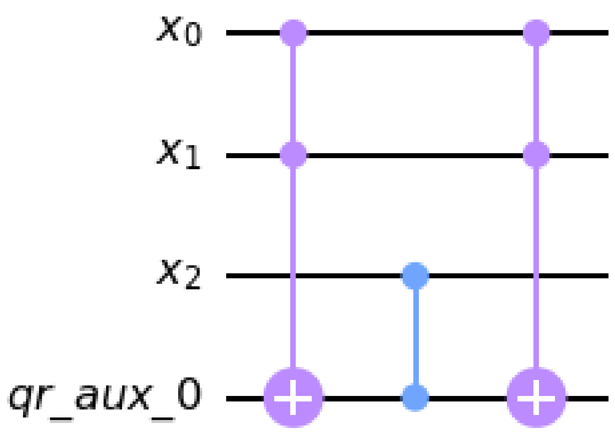
[4]:
```
def decompose_CnZ(circuit, qr_x, qr_a):
    circuit.ccx(qr_x[0],qr_x[1],qr_a[0])

    for i in range(2, x-1):
        circuit.ccx(qr_x[i],qr_a[i-2],qr_a[i-1])

    circuit.cz(qr_a[x-3], qr_x[x-1])

    for i in range(x-2, 1, -1):
        circuit.ccx(qr_x[i],qr_a[i-2],qr_a[i-1])

    circuit.ccx(qr_x[0],qr_x[1],qr_a[0])
```

[5]:
```
qr_a= QuantumRegister(x-2,'qr_aux')
```

[6]:
```
qr_x = QuantumRegister(x, 'x')
```

[7]:
```
qc_help5= QuantumCircuit(qr_x, qr_a)
decompose_CnZ(qc_help5, qr_x, qr_a)

qc_help5.draw(output='mpl')
```

[7]:



[8]:
```
def select_w(circuit, qr_x):
    circuit.x(qr_x[0])
    #circuit.x(qr_x[1])
    #circuit.x(qr_x[2])

def phase_oracle(circuit, qr_x, qr_a):
    select_w(circuit,qr_x)
    decompose_CnZ(circuit, qr_x, qr_a)
    select_w(circuit,qr_x)
```

Diffusor

O Objetivo do difusor é de inverter a fase do input desejado, assim como ampliar a sua amplitude.

[9]:
```
def diffuser(circuit, qr_x, qr_a):
    circuit.h(qr_x)
    circuit.x(qr_x)
    decompose_CnZ(circuit, qr_x, qr_a) #CCCCZ -> CCZ
    circuit.x(qr_x)
    circuit.h(qr_x)
```

Agora vamos calcular o numero de vezes que temos de repetir a execução do oráculo e do difusor. Este valor (x) é determinado por x = $\sqrt{N}$, sendo N o comprimento da lista de entrada (Numero de elementos). Este valor é o numero ideal de ‘‘iterações’’ para ter um bom valor de medição, sem degradação dos resultados.

[10]:
```
import math as m

times= round(m.sqrt(2**x))
print(times)
```

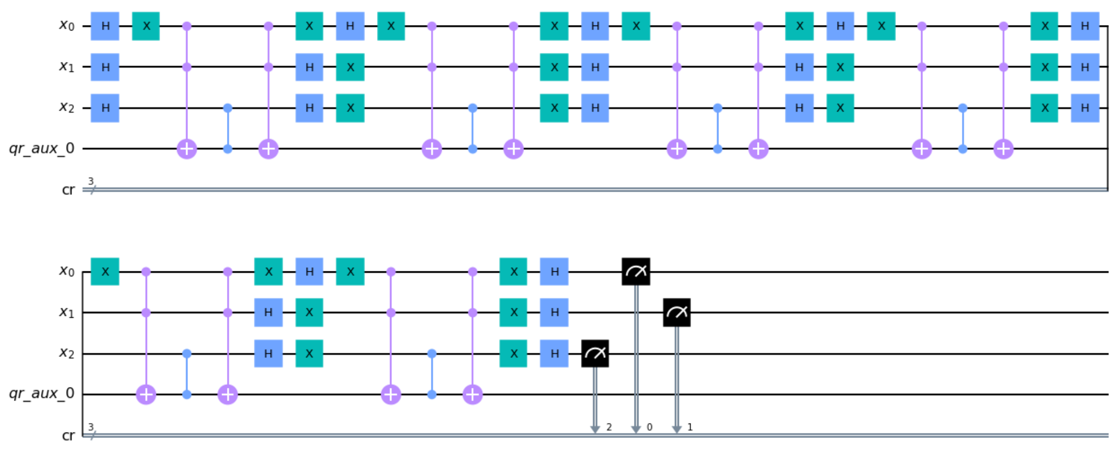
```
3
```

Nesta fase vamos fazer a medição dos qubits.

[11]:
```
backend = Aer.get_backend("qasm_simulator")
```

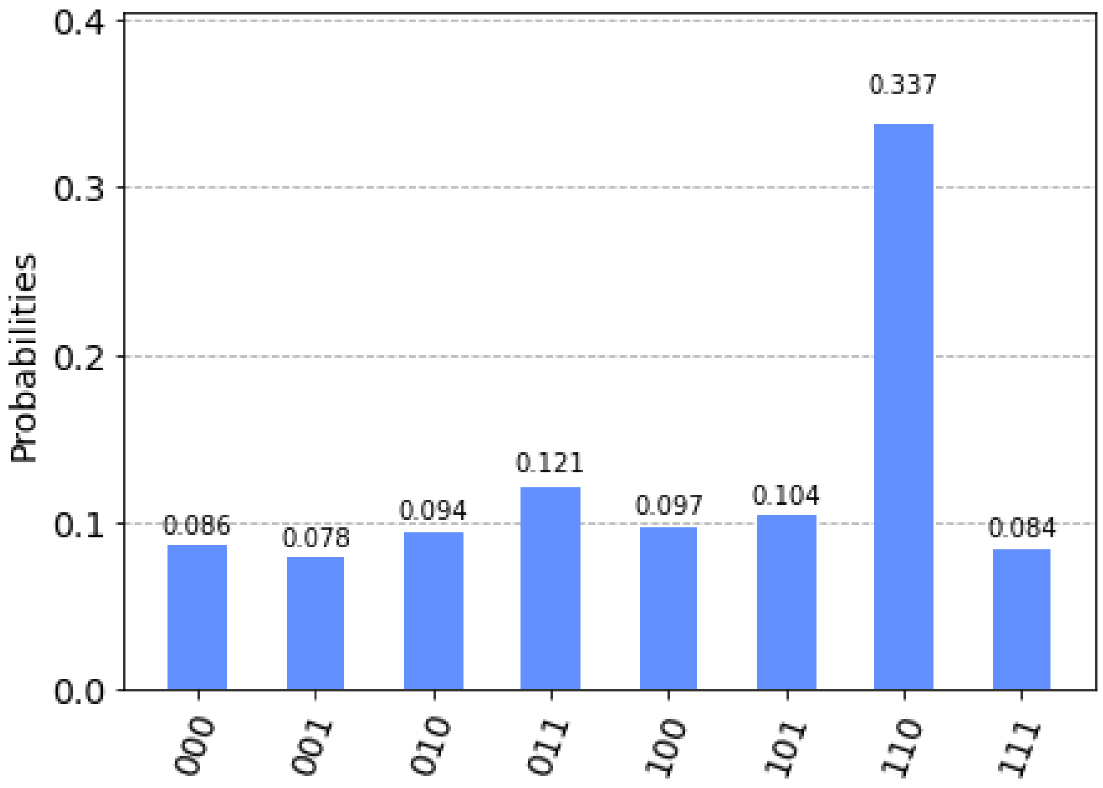
[12]:
```
cr=ClassicalRegister(x,'cr')
qc_Grover= QuantumCircuit(qr_x,qr_a,cr)

# 1.init
qc_Grover.h(qr_x)

# 2.oracle and diffuser

for t in range(times):
    # a)
    phase_oracle(qc_Grover, qr_x,qr_a)
    # b)
    diffuser(qc_Grover,qr_x,qr_a)

# 3.
qc_Grover.measure(qr_x,cr)
qc_Grover.draw(output='mpl')
```

[12]:



[13]:
```
shots=1024
result = execute(qc_Grover, backend, shots=shots).result()
counts_sim = result.get_counts(qc_Grover)
plot_histogram(counts_sim)
```

[13]:



[14]:
```
qc_Grover.depth()
```

[14]:
```
38
```

Nesta fase vamos correr o algoritmo num Computador quântico.

[15]:
```
provider = IBMQ.load_account()
provider.backends()
```

[15]:
```
[<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')>]
```

[16]:
```
# Backend overview
import qiskit.tools.jupyter

%qiskit_backend_overview
```

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

[17]:
```python
from qiskit.tools.monitor import backend_overview, backend_monitor

backend_overview()
```

```
ibmq_manila                 ibmq_quito                  ibmq_belem
-----------                 ----------                  ----------
Num. Qubits:  5             Num. Qubits:  5             Num. Qubits:  5
Pending Jobs: 6             Pending Jobs: 11            Pending Jobs: 2
Least busy:   False         Least busy:   False         Least busy:   False
Operational:  True          Operational:  True          Operational:  True
Avg. T1:      163.8         Avg. T1:      80.3          Avg. T1:      75.9
Avg. T2:      68.2          Avg. T2:      71.2          Avg. T2:      75.6



ibmq_lima                  ibmq_santiago               ibmq_athens
---------                  -------------               -----------
Num. Qubits:  5            Num. Qubits:  5             Num. Qubits:  5
Pending Jobs: 1            Pending Jobs: 8             Pending Jobs: 1
Least busy:   True         Least busy:   False         Least busy:   False
Operational:  True         Operational:  True          Operational:  True
Avg. T1:      62.4         Avg. T1:      110.7         Avg. T1:      96.0
Avg. T2:      50.6         Avg. T2:      100.7         Avg. T2:      104.1



ibmq_armonk                 ibmq_16_melbourne           ibmqx2
-----------                 -----------------           ------
Num. Qubits:  1             Num. Qubits:  15            Num. Qubits:  5
Pending Jobs: 2             Pending Jobs: 11            Pending Jobs: 1
Least busy:   False         Least busy:   False         Least busy:   False
Operational:  True          Operational:  True          Operational:  True
Avg. T1:      169.7         Avg. T1:      54.6          Avg. T1:      58.6
Avg. T2:      261.0         Avg. T2:      53.1          Avg. T2:      35.8
```

[18]:
```python
backend_device = provider.get_backend('ibmq_16_melbourne')
print("Running on: ", backend_device)
```

```
Running on:  ibmq_16_melbourne
```

[19]:
```python
# See backend information
backend_device
```

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

[19]:
```
<IBMQBackend('ibmq_16_melbourne') from IBMQ(hub='ibm-q', group='open',
project='main')>
```

[20]:
```
backend_monitor(backend_device)
```

```
ibmq_16_melbourne
=================
Configuration
-------------
    n_qubits: 15
    operational: True
    status_msg: active
    pending_jobs: 11
    backend_version: 2.3.24
    basis_gates: ['id', 'rz', 'sx', 'x', 'cx']
    local: False
    simulator: False
    memory: True
    allow_q_object: True
    credits_required: True
    coupling_map: [[0, 1], [0, 14], [1, 0], [1, 2], [1, 13], [2, 1], [2, 3], [2,
12], [3, 2], [3, 4], [3, 11], [4, 3], [4, 5], [4, 10], [5, 4], [5, 6], [5, 9],
[6, 5], [6, 8], [7, 8], [8, 6], [8, 7], [8, 9], [9, 5], [9, 8], [9, 10], [10,
4], [10, 9], [10, 11], [11, 3], [11, 10], [11, 12], [12, 2], [12, 11], [12, 13],
[13, 1], [13, 12], [13, 14], [14, 0], [14, 13]]
    n_registers: 1
    quantum_volume: 8
    pulse_num_channels: 9
    max_experiments: 75
    dynamic_reprate_enabled: False
    max_shots: 8192
    supported_instructions: ['cx', 'id', 'delay', 'measure', 'rz', 'sx', 'u1',
'u2', 'u3', 'x']
    processor_type: {'family': 'Canary', 'revision': 1.1}
    dt: 0.2222222222222222
    conditional: False
    description: 15 qubit device
    meas_map: [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]]
    multi_meas_enabled: True
    sample_name: family: Canary, revision: 1.1
    pulse_num_qubits: 3
    input_allowed: ['job']
    url: None
    backend_name: ibmq_16_melbourne
    online_date: 2018-11-06 05:00:00+00:00
    dtm: 0.2222222222222222
    open_pulse: False
    allow_object_storage: True

Qubits [Name / Freq / T1 / T2 / RZ err / SX err / X err / Readout err]
----------------------------------------------------------------------
    Q0 / 5.11472 GHz / 62.93928 us / 99.37162 us / 0.00000 / 0.00043 / 0.00043 /
0.02520
    Q1 / 5.23509 GHz / 48.18623 us / 37.03578 us / 0.00000 / 0.00117 / 0.00117 /
0.03870
    Q2 / 5.03849 GHz / 70.51519 us / 22.31060 us / 0.00000 / 0.00082 / 0.00082 /
0.05760
    Q3 / 4.89444 GHz / 62.71214 us / 19.94602 us / 0.00000 / 0.00050 / 0.00050 /
0.06810
    Q4 / 5.02204 GHz / 50.91581 us / 54.57335 us / 0.00000 / 0.00185 / 0.00185 /
0.05830
    Q5 / 5.07319 GHz / 21.86929 us / 36.54849 us / 0.00000 / 0.00223 / 0.00223 /
0.05270
    Q6 / 4.92950 GHz / 64.25175 us / 83.01192 us / 0.00000 / 0.00087 / 0.00087 /
0.04020
    Q7 / 4.98321 GHz / 34.66348 us / 24.89954 us / 0.00000 / 0.00355 / 0.00355 /
0.03490
    Q8 / 4.75136 GHz / 98.41163 us / 84.55464 us / 0.00000 / 0.00064 / 0.00064 /
0.03840
    Q9 / 4.97357 GHz / 48.00902 us / 57.83415 us / 0.00000 / 0.00158 / 0.00158 /
0.04450
    Q10 / 4.94457 GHz / 47.64211 us / 53.75710 us / 0.00000 / 0.00131 / 0.00131
/ 0.03600
    Q11 / 4.99747 GHz / 55.51833 us / 75.63350 us / 0.00000 / 0.00061 / 0.00061
/ 0.03870
    Q12 / 4.76377 GHz / 74.85701 us / 57.75230 us / 0.00000 / 0.00102 / 0.00102
/ 0.03670
    Q13 / 4.97359 GHz / 31.19523 us / 33.83523 us / 0.00000 / 0.00203 / 0.00203
/ 0.05840
    Q14 / 5.00738 GHz / 47.28912 us / 55.06637 us / 0.00000 / 0.00073 / 0.00073
/ 0.05160

Multi-Qubit Gates [Name / Type / Gate Error]
-------------------------------------------
    cx14_0 / cx / 0.01936
    cx0_14 / cx / 0.01936
    cx14_13 / cx / 0.03627
    cx13_14 / cx / 0.03627
    cx6_8 / cx / 0.02388
    cx8_6 / cx / 0.02388
    cx5_9 / cx / 0.04116
    cx9_5 / cx / 0.04116
    cx4_10 / cx / 0.03160
    cx10_4 / cx / 0.03160
    cx11_3 / cx / 0.03039
```

```
cx3_11 / cx / 0.03039
cx12_2 / cx / 0.05977
cx2_12 / cx / 0.05977
cx13_1 / cx / 0.07569
cx1_13 / cx / 0.07569
cx13_12 / cx / 0.02330
cx12_13 / cx / 0.02330
cx11_12 / cx / 0.01931
cx12_11 / cx / 0.01931
cx10_11 / cx / 0.02509
cx11_10 / cx / 0.02509
cx9_10 / cx / 0.02925
cx10_9 / cx / 0.02925
cx9_8 / cx / 0.04392
cx8_9 / cx / 0.04392
cx7_8 / cx / 0.04080
cx8_7 / cx / 0.04080
cx5_6 / cx / 0.04433
cx6_5 / cx / 0.04433
cx5_4 / cx / 0.03517
cx4_5 / cx / 0.03517
cx4_3 / cx / 0.02980
cx3_4 / cx / 0.02980
cx2_3 / cx / 0.02921
cx3_2 / cx / 0.02921
cx1_2 / cx / 0.01511
cx2_1 / cx / 0.01511
cx1_0 / cx / 0.02084
cx0_1 / cx / 0.02084
```

[21]:
```
%qiskit_job_watcher
```

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

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

[22]:
```
job_r = execute(qc_Grover, backend_device, shots=shots)

jobID_r = job_r.job_id()

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

```
JOB ID: 60bcc8effe8ff1b2ef29f543
```

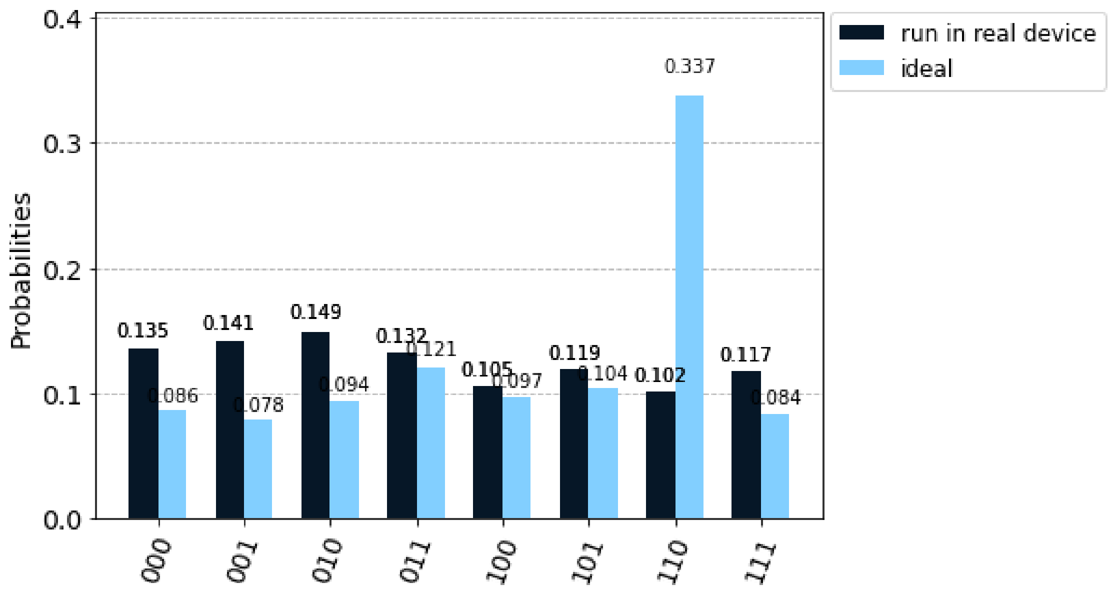
[23]:
```
#ibmq_16_melbourne 4 times the oracle:
job_get=backend_device.retrieve_job("60bb8c6300aded9c926a6e71")
```

```
result_r = job_get.result()
counts_run = result_r.get_counts(qc_Grover)
```

[24]:
```
#ibmq_16_melbourne 1 times the oracle:
job_get=backend_device.retrieve_job("60bb8c6300aded9c926a6e71")

result_r = job_get.result()
counts_run = result_r.get_counts()
```

[25]:
```
plot_histogram([counts_run, counts_sim ], legend=[ 'run in real device',␣
 ↪'ideal'], color=['#061727','#82cfff'])
```

[25]:



2 Ignis

Agora vamos atenuar os erros com o Ignis

[26]:
```
# Import measurement calibration functions
from qiskit.ignis.mitigation.measurement import (complete_meas_cal,␣
 ↪tensored_meas_cal,
                                                 CompleteMeasFitter,␣
 ↪TensoredMeasFitter)
```

Matrizes de calibração Vamos criar uma list com os circuitos de calibração da medição.

Cada circuito cria um estado de base.

Como medimos 3 qubits, precisamos de $2^3 = 8$ circuitos de calibração.

[27]:
```
# Generate the calibration circuits
qr = QuantumRegister(x)

# 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')
```

[28]:
```
state_labels
```

[28]:
```
['000', '001', '010', '011', '100', '101', '110', '111']
```

Computação da matriz de calibração Se não houvesse ruído no dispositivo, a matriz de calibração seria a matriz de identidade 8×8. Uma vez que calculamos esta matriz com um dispositivo quântico real, vai haver algum ruído.

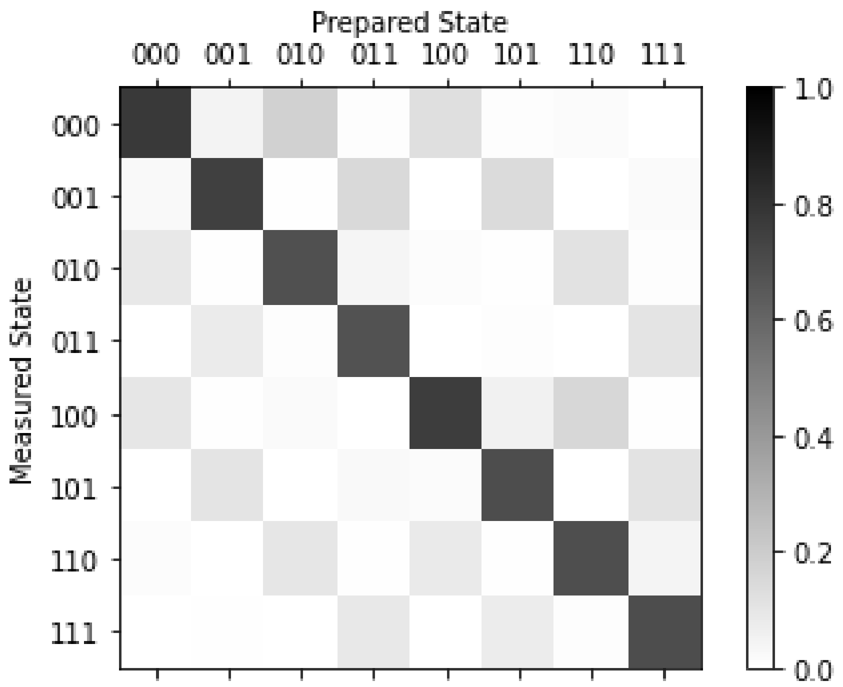
Podíamos fazer este passo com um ruído simulado do Qiskit Aer.

[29]:
```
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: 60bcc8f6bce772159cef948c
```

[30]:
```
job_get=backend_device.retrieve_job("60bb8de3fe8ff1941529e7f6")

cal_results = job_get.result()
```

[31]:
```
meas_fitter = CompleteMeasFitter(cal_results, state_labels, circlabel='mcal')

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



Análise dos resultados

[32]:
```
# What is the measurement fidelity?
print("Average Measurement Fidelity: %f" % meas_fitter.readout_fidelity())
```

```
Average Measurement Fidelity: 0.715454
```

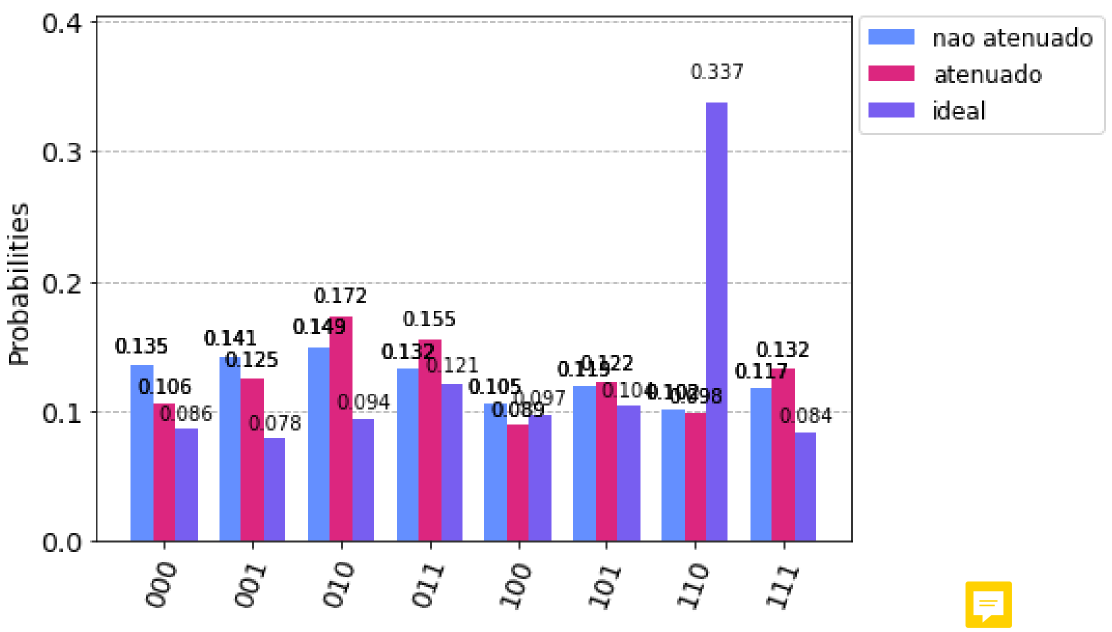
2.0.1 Aplicar a Calibração

Vamos agora aplicar um filtro baseado na matriz de calibração para obter contagens atenuadas.

[33]:
```
# 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()
```

[34]:
```
plot_histogram([counts_run, mitigated_counts, counts_sim], legend=['nao␣
 ↪atenuado', 'atenuado', 'ideal'])
```

[34]:


Probabilities
nao atenuado
atenuado
ideal
0.4
0.3
0.2
0.1
0.0
0.337
0.172
0.155
0.149
0.141
0.135
0.132
0.125
0.122
0.121
0.119
0.117
0.132
0.106
0.105
0.104
0.102
0.098
0.097
0.094
0.089
0.086
0.084
0.078
000
001
010
011
100
101
110
111