日本語訳『Qiskit Textbook』勉強会

5.4章 量子ボリューム（QV）

Kifumi Numata
Oct 14, 2020

5.4章 量子ボリューム（QV）

Quantum Volume（量子ボリューム）とは

Near-term quantum computers 向けの性能指標です。

より高い量子ボリューム(QV)を持つためには、以下が必要です：
- ゲート操作の忠実度が高いこと
- 量子ビットの連結数が多いこと
- ゲートセットが正確にキャリブレーションされていること
- 量子回路書き換えのツールを持つこと

量子ボリューム V_Q は、以下の式で定義されます。

$$\log_2 V_Q = \arg\max_m \min(m, d(m))$$

m: Width(量子ビット数とほぼ同じ意味)

d(m): Depth(QVランダム回路の層)

意味：そのデバイスでQVランダム回路をほぼ正常に計算できる最大のWidthまたはDepth（正方形の回路です：Width=Depth）

Quantum Volume（量子ボリューム）とは

$$\log_2 V_Q = \arg\max_m \min(m, d(m))$$

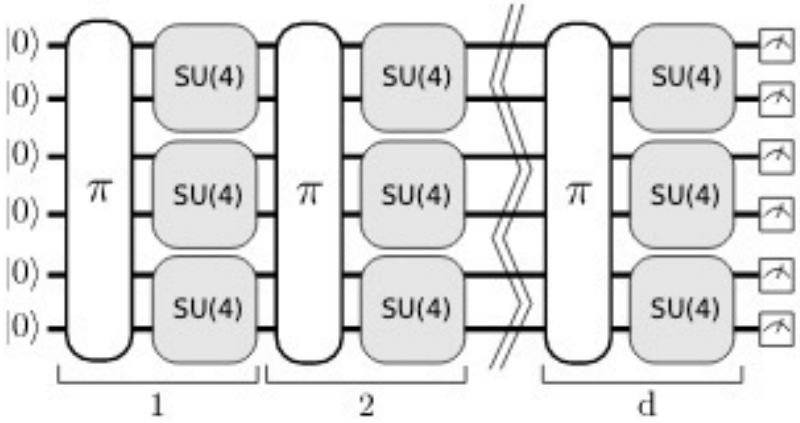
m: Width(量子ビット数とほぼ同じ意味)
d(m): Depth(QVランダム回路の層)
意味：そのデバイスでQVランダム回路をほぼ正常に計算できる最大のWidthまたはDepth
（正方形の回路です：Width=Depth）

- 例えば、QV=64(= 2^6)とは、
  6量子ビットでランダム回路の6-depthまでの回路がほぼ正確に使えるという意味です。
  （物理的に27Qubitのデバイスであってもそのうちの6Qubitがほぼ正確に使える。）
- QVが２倍になるとは、
  6-width, 6-depth が 7-width, 7-depthになるという意味です

- QV=4,000,000(~2^22)とは22量子ビットで22-depth

注： QVのDepthは、u3, cxに分解する前の数です

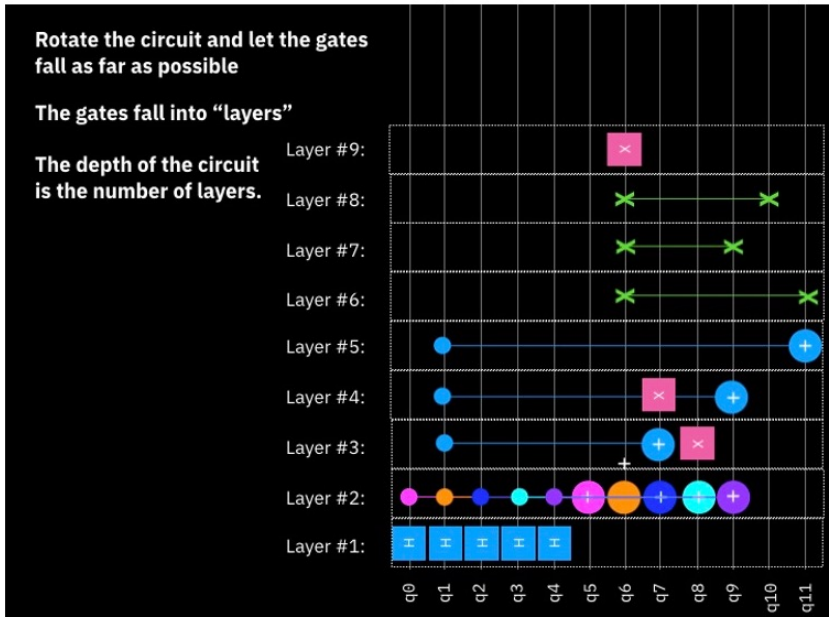
QVランダム回路のDepth



πの部分：各ビットの入れ替え
SU(4): 4x4の行列式が1のユニタリー行列

出典：Andrew W. Cross他 https://arxiv.org/pdf/1811.12926.pdf

u3, cxに分解したときのDepth



https://qiskit.org/documentation/apidoc/circuit.html?highlight=tetris

例：QV32 (2^5)の分解後のDepthは35

Qiskit Advocate 2020のテスト問題: Q2.2

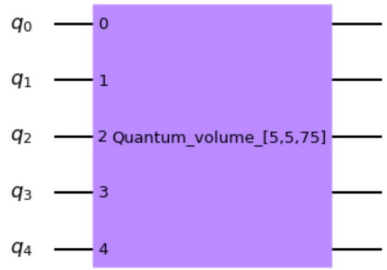
問題) What's the depth of QuantumVolume(num_qubits=5, depth=5) circuit after decomposition into basis gates [u3,cx]?

In [1]:
```
from qiskit.circuit.library import QuantumVolume
```

In [2]:
```
qc = QuantumVolume(num_qubits=5, depth=5)
print('depth =', qc.depth())
qc.draw('mpl')
```

```
depth = 1
```

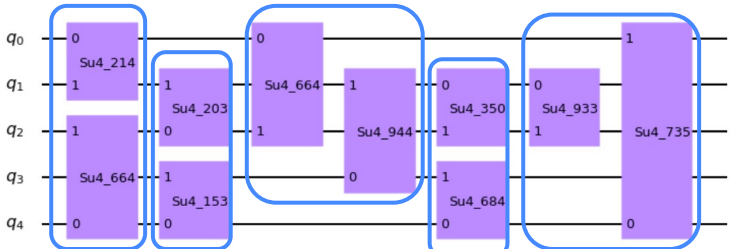
Out[2]:



In [3]:
```
qc_decompose = qc.decompose()
print('depth =', qc_decompose.depth())
qc_decompose.draw('mpl')
```

```
depth = 5
```

Out[3]:

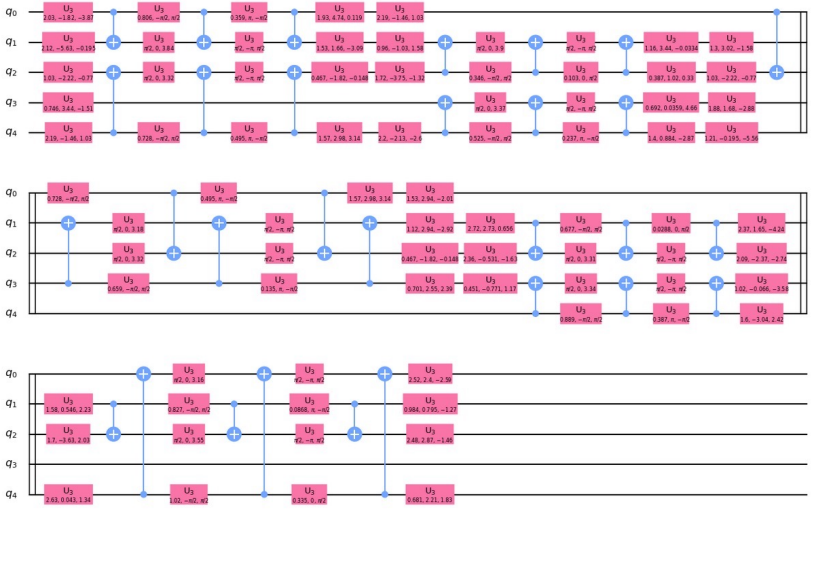


答え) 35

In [4]:
```
qc_decompose2 = qc_decompose.decompose()
print('depth =', qc_decompose2.depth())
qc_decompose2.draw('mpl')
```

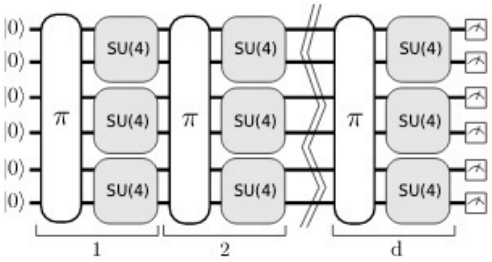
```
depth = 35
```

Out[4]:



Quantum Volume（量子ボリューム）の測定方法

1. モデルとなるQVランダム回路を大量に作る
   （Qiskit Ignisを使う）
   例えば6-qubitのデバイスだったら、3-width, 3-depthから6-width, 6-depthまでの回路をランダムに作る。
   （QVは正方形の回路で定義するため、最大Qubitと最大Depthが同じ。）



πの部分：各ビットの入れ替え
SU(4): 4x4の行列式が1のユニタリー行列

出典：Andrew W. Cross他
https://arxiv.org/pdf/1811.12926.pdf

2. Aerシミュレーターで正解を出しておく
   Heavy outputという結果を得ておく（各Depthごと）。
   （ランダム回路で結果を得る確率は、2/3より大きいことがわかっている。）

3. 実機でQV回路を実行
   Heavy outputを得る確率の平均値を得る（各Depthごと） 。

4. QVを求める
   - Heavy outputの測定確率が2/3より大きいことを確認。（2/3以下のDepthはアウト。）
   - 97.5%以上の精度でHeavy output が2/3より大きい確率で得られているか確認。
     （例えば50回QV回路を実行した時97.5%以上の回路でHOの測定確率が2/3より大きいことを確認。）
   - 以上に合格した最大のDepthに対して、２^(Depth)乗してQVを得る。

Heavy Output Generation (HOG) とは

ランダムな量子回路を実行されたときに、得られる量子ビットの全組み合わせのうち、
測定確率が確率分布の中央値よりも大きい測定結果のことをHeavy Outputという。
たとえば、3-qubitだと000～111まで8通りの出力があるが、
Textbookの最初のランダム回路では、[‘001’, ‘010’, ‘100’, ‘110‘] がHeavy output。

Heavy Outputが測定される確率（この組み合わせの測定確率の和）は、
誤差がない場合、2/3より大きいことが計算から分かっています。
（同じランダム回路を何度も実行し、その出力の統計から確実にHeavy Outputがわかる。）

QV測定では、ランダム量子回路を多数回実行し、
Heavy outputの測定される確率が理想的な場合と比較して97.5%以上の精度があることを確認します。

https://arxiv.org/pdf/1612.05903.pdf

Qiskitでの実行

1．モデルとなるQV回路(ランダム回路)を大量に作る

```
%matplotlib inline
%config InlineBackend.figure_format = 'svg' # 画像をきれいにみせる
import matplotlib.pyplot as plt

#Qiskitのクラスをインポート
import qiskit
from qiskit.providers.aer.noise import NoiseModel
from qiskit.providers.aer.noise.errors.standard_errors \
    import depolarizing_error, thermal_relaxation_error

#QVの機能をインポート
import qiskit.ignis.verification.quantum_volume as qv
```

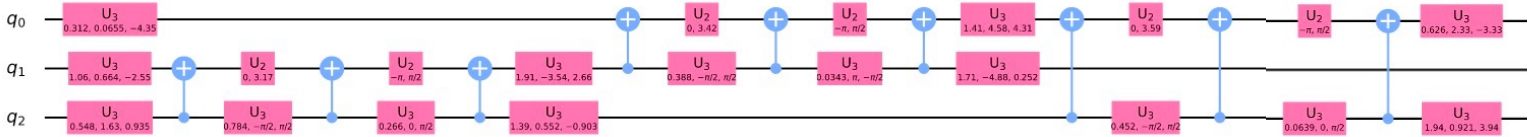
```
# qubit_lists: QV回路を生成するための量子ビットサブセットのリストのリスト
qubit_lists = [[0,1,3],[0,1,3,5],[0,1,3,5,7],[0,1,3,5,7,10]]
# ntrials: サブセットごとに作成するランダム回路の数
ntrials = 50
```

```
qv_circs, qv_circs_nomeas = qv.qv_circuits(qubit_lists, ntrials)
```

Qiskit Ignisを使う

今回は、6-qubitのデバイスを想定しているので、
3-width, 3-depthから
6-width, 6-depthまで
のランダムQV回路を大量に
（50セット）作る。
QVは正方形の回路で定義するため、最大Qubitと最大Depthが同じ。

[0,1,3]のサブセットの回路例（transpile後）



2. AerシミュレーターでQV回路の正解を出しておく

```
#ユニタリー行列はグローバルフェーズではidentity(単位行列)です
backend = qiskit.Aer.get_backend('statevector_simulator')
ideal_results = []
for trial in range(ntrials):
    print('Simulating trial %d'%trial)
    ideal_results.append(qiskit.execute(qv_circs_nomeas[trial], \
                                        backend=backend).result())
```

50セット、実行。

```
Simulating trial 0
Simulating trial 1
```

```
qv_fitter = qv.QVFitter(qubit_lists=qubit_lists)
qv_fitter.add_statevectors(ideal_results)
```

QVフィッターに結果を追加。

```
for qubit_list in qubit_lists:
    l = len(qubit_list)
    print ('qv_depth_'+str(l)+'_trial_0:', \
           qv_fitter._heavy_outputs['qv_depth_'+str(l)+'_trial_0'])
```

Heavy Outputの例を見てみる。
（1セット目のみ）

```
qv_depth_3_trial_0: ['001', '100', '101', '111']
qv_depth_4_trial_0: ['0010', '0011', '0110', '1000', '1010', '1011', '1100', '1110']
qv_depth_5_trial_0: ['00001', '00011', '00100', '00111', '01000', '01001', '01011', '01100', '01111', '10001', '10100', '10111', '11000', '11101', '11110', '11111']
qv_depth_6_trial_0: ['000000', '000001', '000100', '000101', '000111', '010000', '010001', '010011', '010100', '010101', '010110', '010111', '011010', '011101', '011111', '100000',
'100001', '100010', '100110', '100111', '101000', '101001', '101010', '101100', '101101', '110011', '110110', '111010', '111011', '111100', '111101', '111111']
```

```
for qubit_list in qubit_lists:
    l = len(qubit_list)
    print ('qv_depth_'+str(l)+'_trial_0:', \
        qv_fitter._heavy_output_prob_ideal['qv_depth_'+str(l)+'_trial_0'])
```

Heavy Outputの測定確率例を見てみる。
（1セット目のみ）
シミュレーターなので全て
2/3=0.66以上であることが確認できる。

```
qv_depth_3_trial_0: 0.787486996052323
qv_depth_4_trial_0: 0.9554785785151736
qv_depth_5_trial_0: 0.8582030431605264
qv_depth_6_trial_0: 0.8422121051008331
```

3. 実機（今回はノイズありシミュレーター）でQV回路を実行

```
noise_model = NoiseModel()
p1Q = 0.002
p2Q = 0.02
noise_model.add_all_qubit_quantum_error(depolarizing_error(p1Q, 1), 'u2')
noise_model.add_all_qubit_quantum_error(depolarizing_error(2*p1Q, 1), 'u3'
noise_model.add_all_qubit_quantum_error(depolarizing_error(p2Q, 2), 'cx')
```

脱分極エラーでノイズを作る

```
backend = qiskit.Aer.get_backend('qasm_simulator')
basis_gates = ['u1','u2','u3','cx'] # use U,CX for now
shots = 1024
exp_results = []
for trial in range(ntrials):
    print('Running trial %d'%trial)
    exp_results.append(qiskit.execute(qv_circs[trial], \
        basis_gates=basis_gates, \
        backend=backend, \
        noise_model=noise_model, \
        backend_options={'max_parallel_experiments': 0}).result())
```

ノイズありシミュレーターで
50セットのQV回路を実行。

```
qv_fitter.add_data(exp_results)
for qubit_list in qubit_lists:
    l = len(qubit_list)
    #print (qv_fitter._heavy_output_counts)
    print ('qv_depth_'+str(l)+'_trial_0:', \
        qv_fitter._heavy_output_counts['qv_depth_'+str(l)+'_trial_0'])
```

QVフィッターに結果を追加。

Heavy Outputの測定結果を見てみる。
（1セット目のみ）
ショット数1024なので、
683個以上（2/3以上）あれば、ここまではパス。

```
qv_depth_3_trial_0: 756
qv_depth_4_trial_0: 846
qv_depth_5_trial_0: 700
qv_depth_6_trial_0: 652
```

← Depth 6はこの時点でダメ。

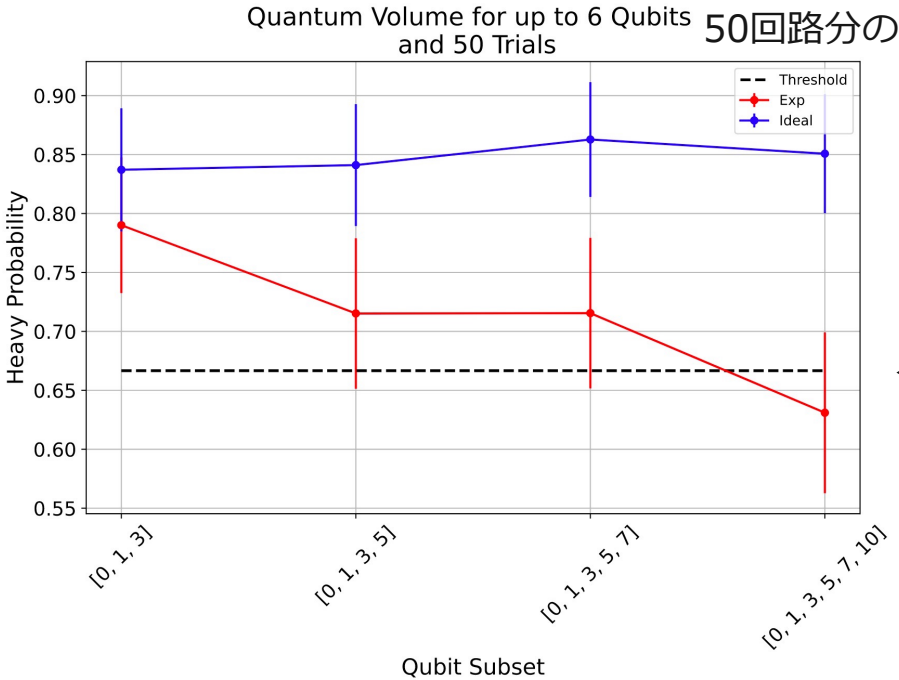
```
plt.figure(figsize=(10, 6))
ax = plt.gca()

# Plot the essence by calling plot_rb_data
qv_fitter.plot_qv_data(ax=ax, show_plt=False)

# Add title and label
ax.set_title('Quantum Volume for up to %d Qubits \n and %d Trials'\
             %(len(qubit_lists[-1]), ntrials), fontsize=18)

plt.show()
```

QVフィッターに貯めた結果をプロット。



50回路分の平均

赤：実機（今回はノイズありシミュレーター）
青：理想的シミュレーター

← 2/3のクライテリア

Depth 6は不合格。

4. QVを求める

```
qv_success_list = qv_fitter.qv_success()
qv_list = qv_fitter.ydata
QV = 1
for qidx, qubit_list in enumerate(qubit_lists):
    if qv_list[0][qidx]>2/3:
        if qv_success_list[qidx][0]:
            print("Width/depth %d greater than 2/3 (%f) with confidence \
                %f (successful). Quantum volume %d"%
                  (len(qubit_list),qv_list[0][qidx],\
                   qv_success_list[qidx][1],\
                   qv_fitter.quantum_volume()[qidx]))
            QV = qv_fitter.quantum_volume()[qidx]
        else:
            print("Width/depth %d greater than 2/3 (%f) with confidence \
                %f (unsuccessful)."%
                  (len(qubit_list),qv_list[0][qidx],\
                   qv_success_list[qidx][1]))
    else:
        print("Width/depth %d less than 2/3 (unsuccessful)."\
              %len(qubit_list))
```

qv_success_list ：各Depthが成功したかどうか（97.5%を超える信頼度で>2/3をT/Fで ）と信頼度を入れる。

qv_list：各Depthの出力確率の平均と標準を入れる。

qidx, qubit_listが[0,0]~[3,3]のとき

出力確率の平均>2/3で
97.5%を超える信頼度で成功していたら、
QVの計算値を出力。

```
Width/depth 3 greater than 2/3 (0.813027) with confidence 0.996028 (successful). Quantum volume 8
Width/depth 4 greater than 2/3 (0.711250) with confidence 0.756673 (unsuccessful).
Width/depth 5 greater than 2/3 (0.716406) with confidence 0.782392 (unsuccessful).
Width/depth 6 less than 2/3 (unsuccessful).
```

```
print ("The Quantum Volume is:", QV)
```

今回はDepth3のみ信頼度0.975を超えていたので、
2^3=8よりQV=8

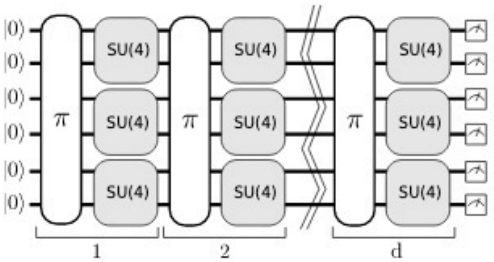
```
The Quantum Volume is: 8
```

まとめ：Quantum Volume

$$\log_2 V_Q = \arg\max_m \min(m, d(m))$$

m: Width(量子ビット数とほぼ同じ意味)、 d(m): Depth(QVランダム回路の層)

意味：（logQVが）QVランダム回路をほぼ正常に計算できる最大のWidthまたはDepth



πの部分：各ビットの入れ替え

SU(4): 4x4の行列式が1のユニタリー行列

出典：Andrew W. Cross他
https://arxiv.org/pdf/1811.12926.pdf

QVの測定の仕方：

1. モデルとなるQV回路(ランダム回路)を大量に作る（Qiskit Ignisを使う）
2. Aerシミュレーターで正解を出しておく
3. 実機でQV回路を実行
4. QVを求める
   - Heavy outputの測定確率が2/3より大きいことを確認。（2/3以下のDepthはアウト。）
   - 97.5%以上の精度でHeavy output が2/3より大きい確率で得られているか確認。
     （例えば50回QV回路を実行した時97.5%以上の回路でHOの測定確率が2/3より大きいことを確認。）
   - 以上に合格した最大のDepthに対して、 2^(Depth)乗してQVを得る。