# Improvements in randomized benchmarking

## qiskit_experiments

Merav Aharoni and Itoko Toshinari

# Agenda

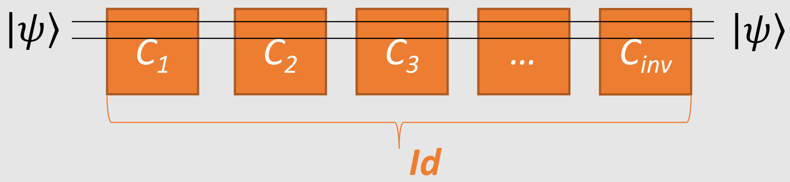
- (Very) short reminder on what randomized benchmarking (RB) is
- The existing algorithm for randomized benchmarking in qiskit_experiments, and its bottlenecks
- Main ideas for performance improvements for 1 and 2 qubits
- Main improvements in the code structure
- Benchmarking results

# Randomized Benchmarking (RB) – brief reminder

**RB is a protocol that provides an estimate of the average error-rate for a set of quantum gate operations**

Consists of the following three stages:

a) Generate RB sequences consisting of random elements from the Clifford group, followed by the Clifford that is the inverse of the random sequence.




b) Run the RB sequences either on the device or on the simulator (with a noise model) and compare to the initial state

c) Get the statistics and fit an exponential decaying curve: $Ap^m+B$
Compute error per Clifford (EPC): $r=(1-p)(d-1)/d \quad (d=2^n)$
From this, compute error per gate.

[1] Easwar Magesan, J. M. Gambetta, and Joseph Emerson, *Robust randomized benchmarking of quantum processes*, ttps://arxiv.org/pdf/1009.3639

[2] Easwar Magesan, Jay M. Gambetta, and Joseph Emerson, *Characterizing Quantum Gates via Randomized Benchmarking*, https://arxiv.org/pdf/1109.6887

https://github.com/Qiskit/qiskit-tutorials/blob/master/qiskit/advanced/ignis/5a_randomized_benchmarking.ipynb

# Current implementation – rough description

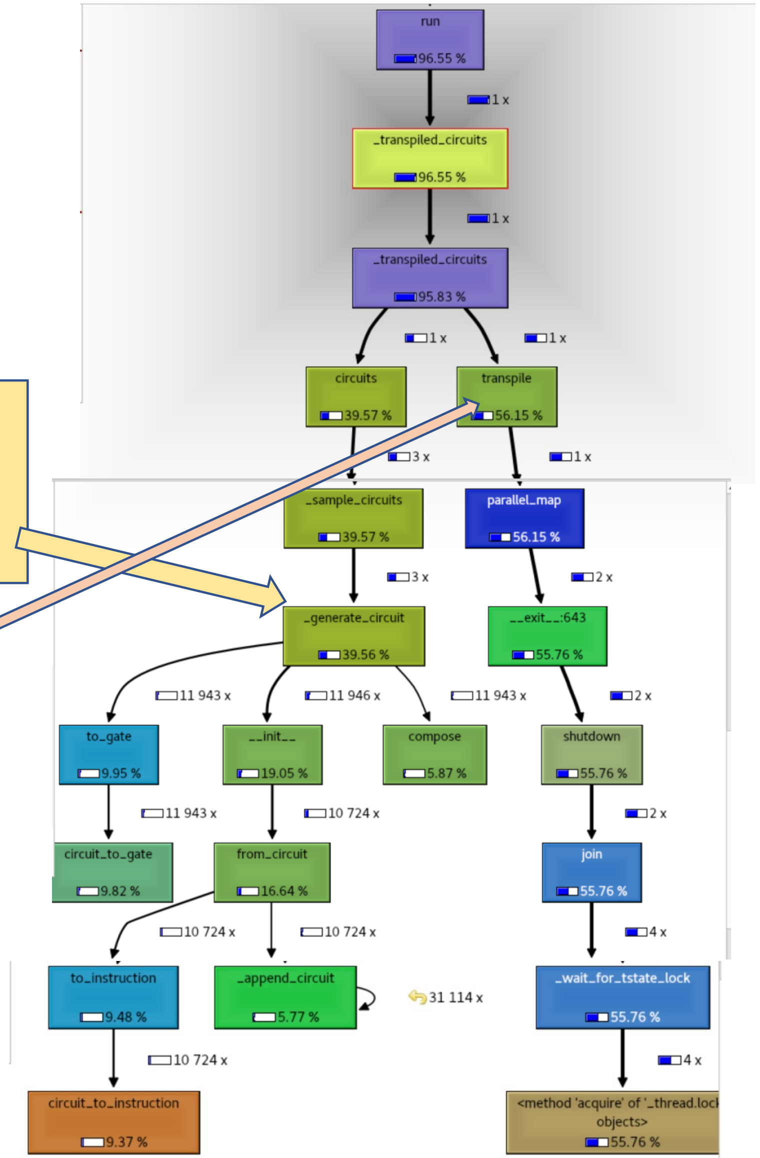
```
list_of_circuits = []
For length in all_lengths
        circuit = QuantumCircuit
        current_clifford = Id
        For i in 1 to length-1
                generate one random_Clifford
                append random_Clifford to circuit
                current_clifford = current_Clifford ∘ random_Clifford
                append barrier
        inverse_clifford = inverse(current_clifford)
        append inverse_clifford to circuit
        append circuit to list_of_circuits

transpile all circuits in list_of_circuits
run on backend
analyze results
```

# Current implementation – bottlenecks

*list_of_circuits* = []
For *length* in *all_lengths*
    *circuit* = *QuantumCircuit*
    *current_clifford* = *Id*
    For *i* in *1* to *length-1*
        generate one *random_Clifford*
        append *random_Clifford* to *circuit*
        *current_clifford* = *current_Clifford* ∘ *random_Clifford*
        append barrier
    *inverse_clifford* = inverse(*current_clifford*)
    append *inverse_clifford* to *circuit*
    append *circuit* to *list_of_circuits*

transpile all circuits in *list_of_circuits*
run on backend
analyze results



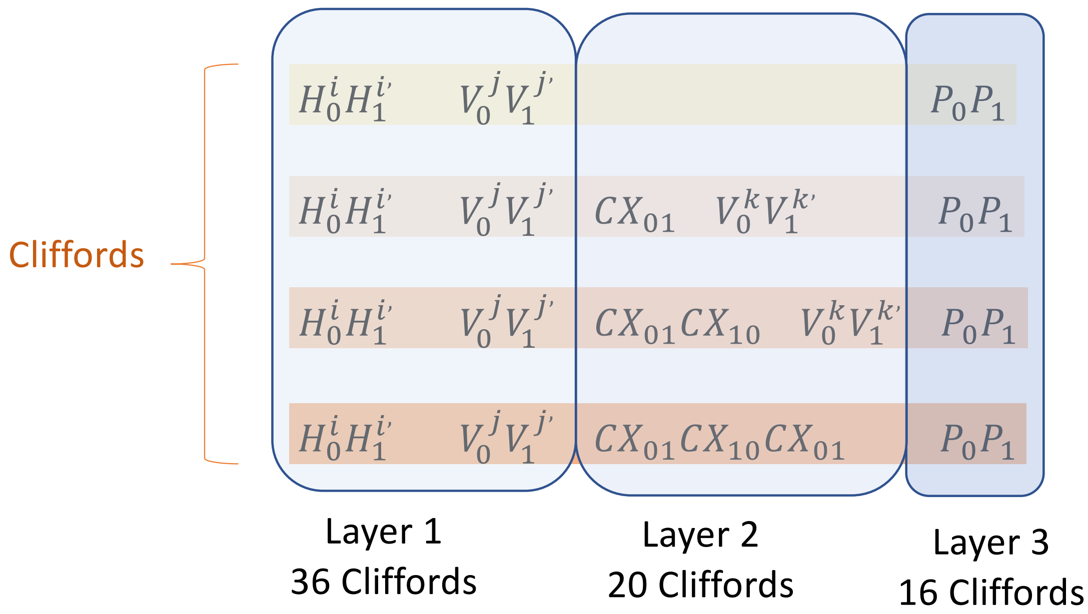
# New implementation

Three main ideas:

1. Eliminate transpilation of the entire circuit
2. Speed up operations on Cliffords
3. Reasonable storage space

# 1. Eliminate transpilation of the entire circuit

- Build the Cliffords in 3 layers
- Transpile these Cliffords in advance




$$i, i' \in \{0,1\}, \quad j, j', k, k' \in \{0,1,2\}, \quad P_0, P_1 \in \{I, X, Y, Z\}$$
$$V = S^{\dagger} H$$

# 2. Speed up operations on Cliffords (create, compose and inverse)

Create an isomorphism between the group of Cliffords and a group of integers.

Clifford1 Clifford2 … Clifford_*n*

$\updownarrow$ $\updownarrow$ $\updownarrow$

*1* *2* … *n*

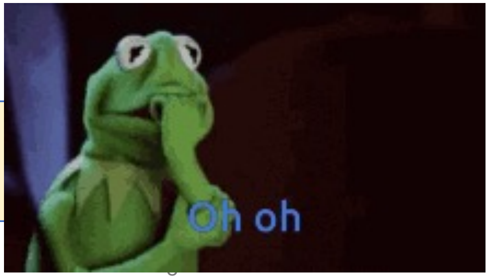
- Clifford_*j* ∘ Clifford_*k* = Clifford_*m* $\longleftrightarrow$ *j* ∘ *k* = *m*
- inverse(Clifford_*j*) = Clifford_*k* $\longleftrightarrow$ Inverse(*j*) = *k*

Perform all group operations on the group of integers instead of on the group of Cliffords.

Store a list of all the results of compose. Access the result of $j \circ k$ according to the indices *j* and *k*.
Store a list of all the results of inverse. Access the inverse of *j* according to the index of *j*.

These lists are constant and are generated once for all RB experiments.

Note – the size of the compose list is $n^2$, where $n$ is the size of the Clifford group.
For 2 qubits $11520^2$ = 132,710,400



# 3. Reduce storage space for compose results

Recall that the Clifford Group for 1 qubit is generated *by clifford_single_gates_1q = {S, H}.*
For 2 qubits, it is generated by *clifford_single_gates_2q = {S(0), S(1), H(0), H(1), CX(0,1), CX(1,0)}.*

In other words, we can decompose every Clifford into these gates.

So instead of computing $C_1 \circ C_2$, *we can decompose* $C_2$ *into* $C_2 = C_2^0 \circ C_2^1 \circ \ldots \circ C_2^k$
Where $C_2^i \in$ *clifford_single_gates,* $0 \leq i \leq k$
And then compute $C_1 \circ C_2^0 \circ C_2^1 \circ \ldots \circ C_2^k$.

This means we can reduce the composition table

- For 1 qubit - from 24 X 24 to 24 X 2.
- For 2 qubits – from 11520 X 11520 to 11520 X 6 = 69,120*.

* In practice for convenience, we store the entries for 21 gates, for a total storage of 241,920 entries

# New algorithm – rough description

```
transpiled_Cliffords = [transpile(layer1, layer2, layer3)]
list_of_circuits = []
For length in all_lengths
        circuit = QuantumCircuit
        current_clifford_num = 0
        For i in 1 to length-1
            generate a random triplet of integers c1, c2, c3 from a random num in [0,…,num_cliffords]
            rand_cliff = convert_to_Clifford(c1 ◦ c2 ◦ c3)
            circuit.append(rand_cliff)
            circuit.append(barrier)
            current_clifford_num =
                current_clifford_num ◦ c1◦ c2◦… ◦ c3    # lookup in list
        inverse_cliff_num = inverse(current_clifford_num)    # lookup in list
        inverse_cliff = transpiled_Cliffords[inverse_cliff_num]
        circuit.append(inverse_cliff)
        append circuit to list_of_circuits

perform trivial placement on physical qubits
run on backend
analyse results
```

Itoko – your turn now!