Abraxas

Any-to-Any Transpilation of Quantum Circuits

Qiskit

NVIDIA
CUDA QUANTUM

PENNYLANE

Cirq

QUANTINUUM

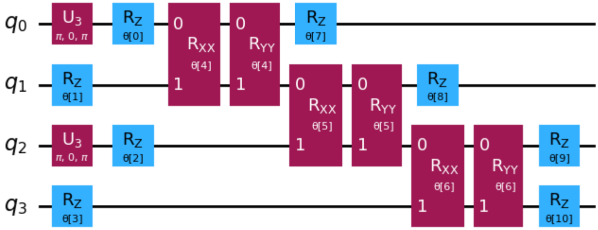
rigetti

Amazon Braket

Example: Qiskit to Bracket Transpilation

```python
qc = QuantumCircuit(4)
qc.x([0, 2])
ep = ExcitationPreserving(4, 'iswap', 'linear', 1)
qc.compose(ep, inplace=True)
qc = qc.decompose()
```

Figure: Qiskit Excitation Preserving H_2



Example: Qiskit to Bracket Transpilation

```python
from abrax import toQasm, toBraket

# convert
qasm = toQasm(qc)
qc = toBraket(qasm)

# run
from braket.devices import LocalSimulator
result = LocalSimulator().run(bell).result()
print(result.measurement_counts)
```

Other Frameworks

```
from pennylane import device
from abrax import *

dev = device("default.qubit", wires=2)

qc = toPenny(QASM, dev)
# qc = toQiskit(QASM)
# qc = toCirq(QASM)
# qc = toTket(QASM)
# qc = toBraket(QASM)
# kern, qubits, thetas = toCudaq(QASM)
# qc = toQuil(QASM)
```

Lower level representations and utilities

```
from abrax import *

qasm2 = toQasm(qc)
qasm3 = toQasm3(qc)
quil = toQuil(qc)
qir = toQir(qc)
string = draw(qc)
```

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[2];
creg meas[2];
h q[0];
cx q[0],q[1];
```

Requirements

- Qiskit
- Numpy
- your respective framework

Ranken

Finding the rank of a subspace

Find the rank of an arbitrary subspace made of up q$d$its as presented by *Zhu, Zhang and Zeng* in *Quantifying subspace entanglement with geometric measures*.

Constructing a state

```
from numpy import sin, cos, pi, eye
from ranken import *

Rn = 3
T = (pi/2)/2
I = eye(Rn)

def PSI(i):
  A = State.create(State.Ket_0, I[i%Rn])
  B = State.create(State.Ket_1, I[(i+1)%Rn])

  coeffs = [cos(T), sin(T)]
  return State.combine([A, B], coeffs)
```

Make a subspace and optionally orthonormalize it

```
from numpy import array

basis = array([PSI(i) for i in range(2)])
proj, proj_perp = Projector(basis, gs=False)

def phi_ik(X):
  qubit = Qdit(2, X[1:5].reshape(2, 2))
  qutrit = Qdit(3, X[5:11].reshape(3, 2))

  return X[0], qubit, qutrit
```

Create the loss function of an arbitrary state in the selected subspace

```
def f(X):
  L, qbit, qtrit = phi_ik(X)
  PHI_rx = normalise(L * kron(qbit, qtrit))
  # Multiple phis can be done with
  # PHI_rx = State.combine(
  #    [phi_rx1, phi_rx2, phi_rx3],
  #    [1, 1, 1]
  # )

  return Loss(PHI_rx, proj_perp)
```

Run the minimization

```
# accepts all args of scipy.optimize.minimiz
size = (2*D + 1)*(r - 1)
rank = minima(f, rand(R_MAX, size), tries=2)
print(f'E_r(={T}) = {rank}')
```

Requirements

- Numpy
- Scipy