Qiskit compatibility with Parameters

Rafał Pracht, Jesús Sistos

Mentored by Nick Bronn

Description

The ***Parameter*** and ***ParameterExpression*** classes in ***Qiskit*** allow for the construction and manipulation of circuits with symbolic expressions, which can be bound to numeric values later, allowing flexibility across many ***Qiskit*** modules. This is especially useful in (classical) computationally intensive steps where the same procedure would need to be repeated for each value of the parameter. However, many ***Qiskit*** modules have limited Parameter support, which limits their flexibility, and this can be improved by appropriate type-checking and parsing

Qiskit

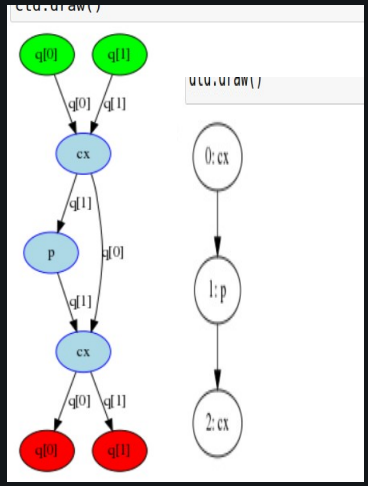
1. Passing unbound parameters through Template Optimization Pass

```
#phi = 0.13
#phi = Parameter('φ')
phi = Parameter('$\\phi$')


qc = QuantumCircuit(2)
qc.cx(0,1)
qc.p(2*phi, 1)
qc.cx(0,1)
print('Original circuit:')
qc.draw(output='mpl')
```

```
Original circuit:
```

Template matching makes use of the circuit's DAG (Directed Acyclic Graph)



Sympy fails when parsing LaTeX names

```
or: Error from parse_expr with transformed code: "$\\Sy

sformed code: "$\\Symbol ('phi' )$"
```

Qiskit

Finding the bug

Parsing occurs in
qiskit/transpiler/passes/optimization/template_matching/template_substitution.py

mpy_parser.py | passmanager.py | template_substitution.py | relational.py | parameterexpression.py | runningpassmanager.py | template_optimization.py

```
491
492         for idx, t_idx in enumerate(template_sublist):
493             qc_idx = circuit_sublist[idx]
494             circuit_params += self.circuit_dag_dep.get_node(qc_idx).op.params
495             template_params += template_dag_dep.get_node(t_idx).op.params
496
497         # Create the fake binding dict and check
498         equations, circ_dict, temp_symbols, sol, fake_bind = [], {}, {}, {}, {}
499         for t_idx, temp_params in enumerate(template_params):
500             if isinstance(temp_params, ParameterExpression):
501                 circ_param_str = str(circuit_params[t_idx])
502                 equations.append(sym.Eq(parse_expr(str(temp_params)), parse_expr(circ_param_str)))
503
504                 for param in temp_params.parameters:
505                     temp_symbols[param] = sym.Symbol(str(param))
506
```

Making the fix

Qiskit

Current code uses a custom parser.

We can use Sympy's tools to do it in a compatible way.

qiskit/transpiler/passes/optimization/template_matching/template_substitution.py

```
@@ -498,8 +498,11 @@ def _attempt_bind(self, template_sublist, circuit_sublist):
            equations, circ_dict, temp_symbols, sol, fake_bind = [], {}, {}, {}, {}
            for t_idx, temp_params in enumerate(template_params):
                if isinstance(temp_params, ParameterExpression):
-                   circ_param_str = str(circuit_params[t_idx])
-                   equations.append(sym.Eq(parse_expr(str(temp_params)), parse_expr(circ_param_str)))
+                   if isinstance(circuit_params[t_idx], ParameterExpression):
+                       circ_param_sym = circuit_params[t_idx].get_expr()
+                   else:
+                       circ_param_sym = parse_expr(str(circuit_params[t_idx]))
+                   equations.append(sym.Eq(temp_params.get_expr(), circ_param_sym))
```

These changes have already been added to the main pull request in Terra's repository.

2. Parameters in Qiskit Nature: Second Quantization Operators

Operator classes directly block the usage of Parameters

$$H = \mu \sum_{i=1}^{2} a_i^\dagger a_i$$

```
# how do we define parameters?

#mu = 0.13
mu = Parameter('μ')
#mu = Parameter('$\\mu')

h = mu*sum(
    FermionicOp(label) for label in ['IN', 'NI'])
```

```
def mul(self, other: complex) -> "FermionicOp":
    if not isinstance(other, (int, float, complex)):
        raise TypeError(
            f"Unsupported operand type(s) for *: 'FermionicOp'
        )
```

What if we ‘bypass’ this restriction?

```
mapper = JordanWignerMapper()
converter = QubitConverter(mapper=mapper)

h_pauli = converter.convert(h)
```

Qiskit

State of the problem

Qiskit

Usage of Numpy limits versatility of parameters

sparse_pauli_op.py M

qiskit > quantum_info > operators > symplectic > sparse_pauli_op.py > SparsePauliOp > __repr__

```
58          """
59          if isinstance(data, SparsePauliOp):
60              pauli_list = data._pauli_list
61              coeffs = data._coeffs
62          else:
63              pauli_list = PauliList(data)
64              if coeffs is None:
65                  coeffs = np.ones(pauli_list.size, dtype=complex)
66          # Initialize PauliList
67          self._pauli_list = PauliList.from_symplectic(pauli_list.z, pauli_list.x)
68
69          # Initialize Coeffs
70          #import pdb; pdb.set_trace()
71          self._coeffs = np.asarray((-1j) ** pauli_list.phase * coeffs, dtype=complex)
72          if self._coeffs.shape != (self._pauli_list.size,):
73              raise QiskitError(
74                  "coeff vector is incorrect shape for number"
75                  " of Paulis {} != {}".format(self._coeffs.shape, self._pauli_list.size)
76              )
```

Future challenges

Qiskit

Template Optimization and inverses of parametrized gates.

Qiskit Opflow parameter management.

Hardware transpiling of custom VQE Ansatz circuits.