Qiskit compatibility with Parameters

Rafał Pracht, Jesús Sistos

Mentored by Nick Bronn

Description

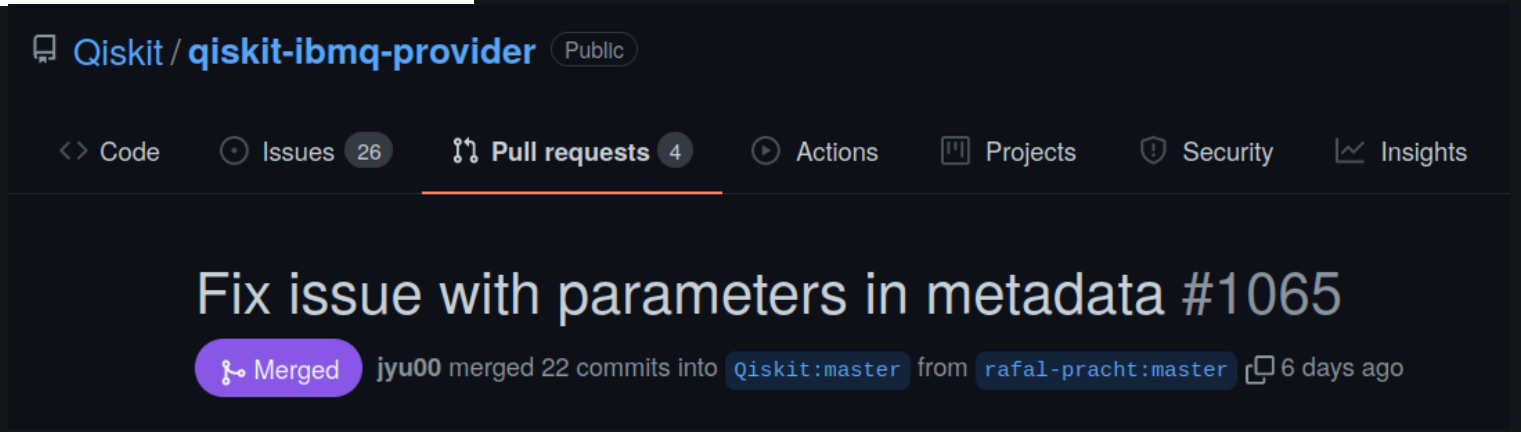
The ***Parameter*** and ***ParameterExpression*** classes in ***Qiskit*** allow for the construction and manipulation of circuits with symbolic expressions, which can be bound to numeric values later, allowing flexibility across many ***Qiskit*** modules. This is especially useful in (classical) computationally 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. Parameter in job metadata

```
qr = QuantumRegister(1)
cr = ClassicalRegister(1)
my_circ_str = 'test_metadata'
my_circ = QuantumCircuit(qr, cr, name=my_circ_str, metadata={Parameter('φ'): 0.2})
my_circ.x(0)
my_circ.y(0)
```

```
job = backend.run(my_circ, shots=1024)
```

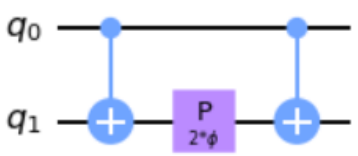


2. Template matching for parameters with LaTeX name

```
#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:
```




```
pass_ = TemplateOptimization(**rzx_templates.rzx_templates(['zz2']))
qc_cz = PassManager(pass_).run(qc)
print('ZX based circuit:')

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

Fix template matching for parameters with LaTeX name. #1

Merged **nbronn** merged 1 commit into `nbronn:template-param-expression` from `rafal-pracht:template-param-expression` on Oct 6

Added Parameter Expression functions for Template Optimization pass #6899

Open **nbronn** wants to merge 78 commits into `Qiskit:main` from `nbronn:template-param-expression`

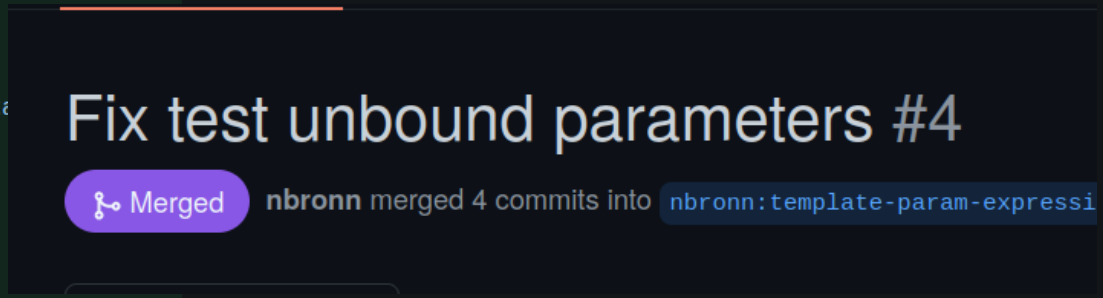
Qiskit

3. Check in UnitaryGate if gate is unitary for unbound parameters.

```
# Check if there is any unbound parameter in array
try:
    # Convert to numpy array in case not already an array
    data = numpy.array(data, dtype=complex)

    # Check input is unitary
    if not is_unitary_matrix(data):
        raise ExtensionError("Input matrix is not unita
except ParameterTypeError:
    from sympy.physics.quantum import Dagger
    matrix, matrix_dim = to_sympy_matrix(data)
    iden = sp.sympify(matrix * Dagger(matrix))

    if iden != sp.eye(matrix_dim):
        # There may be a case when there is still a parameter and
        # we are not quite sure if this is unitary or not.
        # But just in case we throw exception :)
        raise ExtensionError("Input matrix is not unitary.")
```



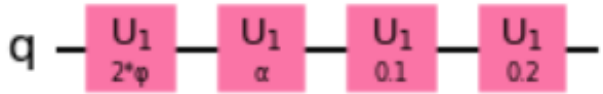
Qiskit

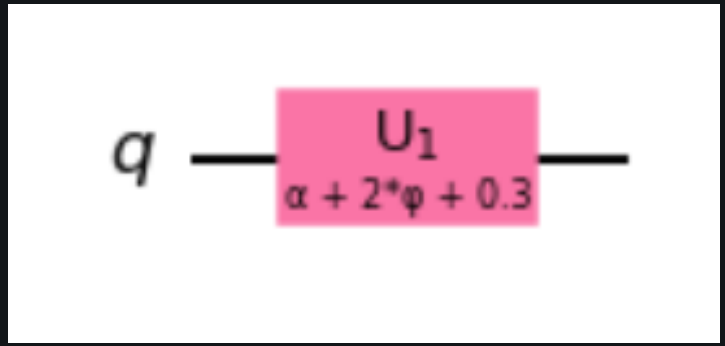
4. Parameters in transpiler.

```
phi = Parameter('φ')
alpha = Parameter('α')

qc = QuantumCircuit(1)
qc.u1(2*phi, 0)
qc.u1(alpha, 0)
qc.u1(0.1, 0)
qc.u1(0.2, 0)
print('Original circuit:')
qc.draw(output='mpl')
```

Original circuit:





Merge u1, u2 and p gate together if they contain parameters. #7309

Open rafal-pracht wants to merge 8 commits into Qiskit:main from rafal-pracht:main

Qiskit

4. Parameterized hamiltonians in Qiskit Nature

Parameter handling in SparsePauliOp #7215

Open **jsistos** wants to merge 12 commits into `Qiskit:main` from `jsistos:operator-parameters`

```
class SparsePauliOp(LinearOp):
@@ -88,7 +89,12 @@ def __init__(self, data, coeffs=None, *, ignore_pauli_pha
        if coeffs is None:
            coeffs = np.ones(pauli_list.size, dtype=complex)
        else:
-           coeffs = np.array(coeffs, copy=copy, dtype=complex)
+           try:
+               coeffs = np.array(coeffs, copy=copy, dtype=complex)
+           except TypeError:
+               # Initialize as array of objects if there are parameters.
+               # This is generally avoided since it makes numpy slower.
+               coeffs = np.array(coeffs, copy=copy, dtype=object)
```

```
def __eq__(self, other):
    """Check if two SparsePauliOp operators are equal"""
    return (
        super().__eq__(other)
        and np.allclose(self.coeffs, other.coeffs)
        and self.paulis == other.paulis
    )
    close_coeffs = []
    for i in range(self.coeffs.shape[0]):
        # Check for Parameters separately
        if isinstance(self.coeffs[i], ParameterExpression):
            close_coeffs.append(self._coeffs[i] == other._coeffs[i])
        else:
            close_coeffs.append(np.isclose(self.coeffs[i], other.coeffs[i]))

    return super().__eq__(other) and np.all(close_coeffs) and self.paulis == other.paulis
```

Qiskit

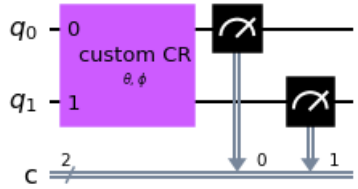
6. Custom parameterized gates in VQE

```
phi = Parameter('$\\phi$')
theta = Parameter('$\\theta$')

qc3 = QuantumCircuit(2, 2)
cr_gate = Gate('custom CR', 2, [theta, phi])
#cr_gate = Gate('custom CR', 2, [])

qc3.append(cr_gate, [0, 1])
qc3.measure([0, 1], [0, 1])

qc3.draw('mpl')
```




```
def visit_ParametricPulse(self, node: ParametricPulse):
    """Assign parameters to ``ParametricPulse`` object."""
    if node.is_parameterized():
        new_parameters = {}
        for op, op_value in node.parameters.items():
            if isinstance(op_value, ParameterExpression):
                op_value = self._assign_parameter_expression(op_value)
            if isinstance(op_value, (int, float, complex)) and  op == 'duration':
                if not np.isclose(op_value % 16, 0):
                    import warnings
                    warnings.warn(
                        f"After parameter assignment, duration was unschedulable ({op_value})."
                        "Rounding down to the nearest multiple of 16.",
                        RuntimeWarning
                    )
                new_parameters[op] = int(op_value - op_value % 16)
            else:
                new_parameters[op] = op_value

        return node.__class__(**new_parameters, name=node.name)

    return node
```

Qiskit