Qiskit Oxide

Matthew Treinish

June 26, 2024

Qiskit data model in 1.0

- QuantumCircuit backed by Rust.
- Started distinguishing between data at rest and data in-use.
- DAGCircuit built using Rustworkx which is a Python graph library implemented in Rust.
- Heavy computation in transpiler passes (and other components) implemented as standalone functions in Rust.

QuantumCircuit Example in 1.0.

```
from math import pi
from qiskit.circuit import QuantumCircuit

qc = QuantumCircuit(2)
qc.rz(pi / 2, 0)
qc.sx(0)
qc.rz(pi / 2, 0)
qc.cx(0, 1)
```

QuantumCircuit Example in 1.0.

```
from math import pi
from qiskit.circuit import QuantumCircuit, CircuitInstruction
from qiskit.circuit.library import RZGate, SXGate, CXGate

qc = QuantumCircuit(2)
qc._append(CircuitInstruction(RZGate(pi / 2), qubits=(qc.qubits[0],)))
qc._append(CircuitInstruction(SXGate(), qubits=(qc.qubits[0],)))
qc._append(CircuitInstruction(RZGate(pi / 2), qubits=(qc.qubits[0],)))
qc._append(CircuitInstruction(CXGate(), qubits=(qc.qubits[0], qc.qubits[1])))
```

QuantumCircuit Example in 1.0.

```rust
use pyo3::prelude::*;
use pyo3::types::PyTuple;

#[pyclass(
    freelist = 20,
    sequence,
    get_all,
    module = "qiskit._accelerate.circuit"
)]
#[derive(Clone, Debug)]
pub struct CircuitInstruction {
    /// The logical operation that this instruction represents an execution of.
    pub operation: PyObject,
    /// A sequence of the qubits that the operation is applied to.
    pub qubits: Py<PyTuple>,
    /// A sequence of the classical bits that this operation reads from or writes
    pub clbits: Py<PyTuple>,
}
```

QuantumCircuit Example in 1.0.

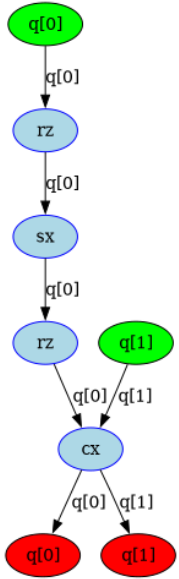
```rust
use pyo3::prelude::*;

pub type IndexType = u32;

struct PackedInstruction {
    /// The Python-side operation instance.
    op: PyObject,
    /// The index under which the interner has stored `qubits`.
    qubits_id: IndexType,
    /// The index under which the interner has stored `clbits`.
    clbits_id: IndexType,
}
```

Qubit representation compression

Python	Indices	At Rest

`[`
`  (qc.qubits[0], qc.qubits[1]),`
`  (qc.qubits[1],),`
`  (qc.qubits[0],),`
`  (qc.qubits[0], qc.qubits[1]),`
`  (qc.qubits[1], qc.qubits[0]),`
`  (qc.qubits[0], qc.qubits[1]),`
`]`
	`[`
	`  (0,1),`
	`  (1,),`
	`  (0,),`
	`  (0, 1),`
	`  (1, 0),`
	`  (0, 1),`
	`]`
		`[`
		`  (0, 1),`
		`  (1,),`
		`  (0,),`
		`  (1, 0),`
		`]`
		`[`
		`  0,`
		`  1,`
		`  2,`
		`  0,`
		`  3,`
		`  0,`
		`]`

Transpiler pass example in 1.0



Transpiler pass example in 1.0

```
class SabreLayout(TransformationPass):

    ...

    def run(self, dag):
        sabre_dag = build_sabre_dag(dag)
        neighbor_table, dist_matrix = build_backend_repr(self.target)
        result = sabre_layout_and_routing(
            sabre_dag,
            neighbor_table,
            dist_matrix,
            ...
        )
        return build_output_dag(result)
```

Transpiler pass example in 1.0

```rust
#[pyfunction]
pub fn sabre_layout_and_routing(
    py: Python,
    dag: &SabreDAG,
    neighbor_table: &NeighborTable,
    distance_matrix: PyReadonlyArray2<f64>,
    ...
) -> (NLayout, SwapMap) {
    ...
}
```

```rust
#[pyclass(module = "qiskit._accelerate.sabre")]
#[derive(Clone, Debug)]
pub struct SabreDAG {
    pub num_qubits: usize,
    pub num_clbits: usize,
    pub dag: DiGraph<DAGNode, ()>,
    ...
}
```

Plan working towards 2.0

- Add representation to Rust for full data model. Rust objects for:
  - Gates
  - DAGCircuit
  - Target
  - Observables
- Rust API still private/internal not exposed to end users
- Optimize for the common path but still support custom workflows
- Expand distinction of data at rest and in-use
- Update transpiler passes to operate solely in Rust domain

QuantumCircuit Example in 1.2.0

```
from math import pi
from qiskit.circuit import QuantumCircuit

qc = QuantumCircuit(2)
qc.rz(pi / 2, 0)
qc.sx(0)
qc.rz(pi / 2, 0)
qc.cx(0, 1)
```

QuantumCircuit Example in 1.2.0

```
from math import pi
from qiskit.circuit import QuantumCircuit, CircuitInstruction
from qiskit._accelerate.circuit import StandardGate

qc = QuantumCircuit(2)
qc._append(
    CircuitInstruction(StandardGate.RZGate, params=[pi / 2], qubits=(qc.qubits[0]
)
qc._append(
    CircuitInstruction(StandardGate.SXGate, qubits=(qc.qubits[0],))
)
qc._append(
    CircuitInstruction(StandardGate.RZGate, params=[pi / 2], qubits=(qc.qubits[0]
)
qc._append(
    CircuitInstruction(StandardGate.CXGate, qubits=(qc.qubits[0], qc.qubits[1]))
)
```

QuantumCircuit Example in 1.2.0

```rust
#[pyclass(freelist = 20, sequence, module = "qiskit._accelerate.circuit")]
#[derive(Clone, Debug)]
pub struct CircuitInstruction {
    pub operation: OperationType,
    /// A sequence of the qubits that the operation is applied to.
    #[pyo3(get)]
    pub qubits: Py<PyTuple>,
    /// A sequence of the classical bits that this operation reads from or writes to.
    #[pyo3(get)]
    pub clbits: Py<PyTuple>,
    pub params: SmallVec<[Param; 3]>,
    pub extra_attrs: Option<Box<ExtraInstructionAttributes>>,
}

pub struct ExtraInstructionAttributes {
    pub label: Option<String>,
    pub duration: Option<PyObject>,
    pub unit: Option<String>,
    pub condition: Option<PyObject>,
}
```

QuantumCircuit Example in 1.2.0

```
pub enum OperationType {
    Standard(StandardGate),
    Instruction(PyInstruction),
    Gate(PyGate),
    Operation(PyOperation),
}
```

QuantumCircuit Example in 1.2.0

```rust
pub enum StandardGate {
    RZGate = 0
    SXGate = 1,
    CXGate = 2,
}

impl Operation for StandardGate {
    ...
}

pub trait Operation {
    fn name(&self) -> &str;
    fn num_qubits(&self) -> u32;
    fn matrix(&self, params: &[Param]) -> Option<Array2<Complex64>>;
    fn definition(&self, params: &[Param]) -> Option<CircuitData>;
}
```

QuantumCircuit Example in 1.2.0

```rust
/// This class is used to wrap a Python side Gate that is not in the stand
#[derive(Clone, Debug)]
#[pyclass(freelist = 20, module = "qiskit._accelerate.circuit")]
pub struct PyGate {
    pub qubits: u32,
    pub clbits: u32,
    pub params: u32,
    pub op_name: String,
    pub gate: PyObject,
}

impl Operation for PyGate {
    ...
}
```

QuantumCircuit Example in 1.2.0

```rust
pub(crate) struct PackedInstruction {
    /// The Python-side operation instance.
    pub op: OperationType,
    /// The index under which the interner has stored `qubits`.
    pub qubits_id: Index,
    /// The index under which the interner has stored `clbits`.
    pub clbits_id: Index,
    pub params: Index,
    pub extra_attrs: Option<Box<ExtraInstructionAttributes>>,
}
```

Circuit Synthesis in mostly Rust

```python
from qiskit.circuit.quantumcircuit import QuantumCircuit
from qiskit.synthesis.permutation_utils import _get_ordered_swap

def synth_permutation_basic(pattern: list[int] | np.ndarray[int]) -> QuantumCircu

    num_qubits = len(pattern)
    qc = QuantumCircuit(num_qubits)

    swaps = _get_ordered_swap(pattern)

    for swap in swaps:
        qc.swap(swap[0], swap[1])

    return qc
```

Circuit Synthesis in mostly Rust

```rust
use qiskit_circuit::circuit_data::CircuitData;
use qiskit_circuit::operations::{Param, StandardGate};
use qiskit_circuit::Qubit;
use crate::synthesis::utils;

#[pyfunction]
#[pyo3(signature = (pattern))]
pub fn _synth_permutation_basic(py: Python, pattern: PyArrayLike1<i64>) -> PyResult<CircuitData> {
    let view = pattern.as_array();
    let num_qubits = view.len();
    CircuitData::from_standard_gates(
        py,
        num_qubits as u32,
        utils::get_ordered_swap(&view).iter().map(|(i, j)| {
            (
                StandardGate::SwapGate,
                smallvec![],
                smallvec![Qubit(*i as u32), Qubit(*j as u32)],
            )
        }),
        Param::Float(0.0),
    )
}
```

Circuit Synthesis in mostly Rust

```python
from qiskit.circuit.quantumcircuit import QuantumCircuit
from qiskit._accelerate.synthesis.permutation import _synth_permutation_bas


def synth_permutation_basic(
    pattern: list[int] | np.ndarray[int],
) -> QuantumCircuit:
    return QuantumCircuit._from_circuit_data(
        _synth_permutation_basic(pattern)
    )
```

Transpiler pass example in 1.2

```rust
#[pyclass(module = "qiskit._accelerate.circuit")]
#[derive(Clone, Debug)]
pub struct DAGCircuit {
    ...

    dag: StableDiGraph<NodeType, Wire>,
}

#[derive(Clone, Debug)]
enum NodeType {
    QubitIn(Qubit),
    QubitOut(Qubit),
    ClbitIn(Clbit),
    ClbitOut(Clbit),
    Operation(PackedInstruction),
}
```

Transpiler pass example in 1.2

```
from qiskit._accelerate.sabre import sabre_layout_and_routing

class SabreLayout(TransformationPass):

    ...

    def run(self, dag):
        return sabre_layout_and_routing(dag, self.target, ...)
```

Transpiler pass example in 1.2

```rust
#[pyfunction]
pub fn sabre_layout_and_routing(
    py: Python,
    dag: &DAGCircuit,
    target: &TargetNeighborTable,
    heuristic: Heuristic,
    ...
) -> DAGCircuit {
    ...
}
```

Current Status

- Main branch has initial gate and circuit representation in Rust
- Work ongoing to expand usage of Rust circuit representation and improve representation
- DAGCircuit PR in progress: https://github.com/Qiskit/qiskit/pull/12550
- Target PR in progress: https://github.com/Qiskit/qiskit/pull/12292
- WIP Rust observable PR: https://github.com/Qiskit/qiskit/pull/12671

Questions?