# Performance Improvement of SparsePauliOp

Takashi Imamichi (IBM Research – Tokyo)
imamichi@jp.ibm.com

Joint work with Ikko Hamamura

Qiskit Demoday
Oct. 28, 2021

# Overview

- Background
  - Objective: improve performance of Jordan-Wigner transformation of Qiskit nature
    - JW transformation uses SparsePauliOp as a building block
  - SparsePauliOp: Sparse N-qubit operator in a Pauli basis representation
    - Internal data change (qiskit-terra#6826)
      - PauliTable + ndarray (terra 0.18 – stable branch)
      - PauliList + ndarray (terra 0.19 – main branch)
  - We noticed performance regression with the main branch
    - There was an inefficient code in SparsePauliOp.simplify
  - We also vectorized various methods
- PRs
  - Merged
    - Speed-up SparsePauliOp.simplify qiskit-terra#7122
    - Speed-up SparsePauliOp.compose qiskit-terra#7126
    - Speed-up Pauli._from_label qiskit-terra#7145
  - Under review
    - Speed-up SparsePauliOp._add qiskit-terra#7138
    - Speed-up QubitMapper.mode_based_mapping qiskit-nature#397

Microbenchmark

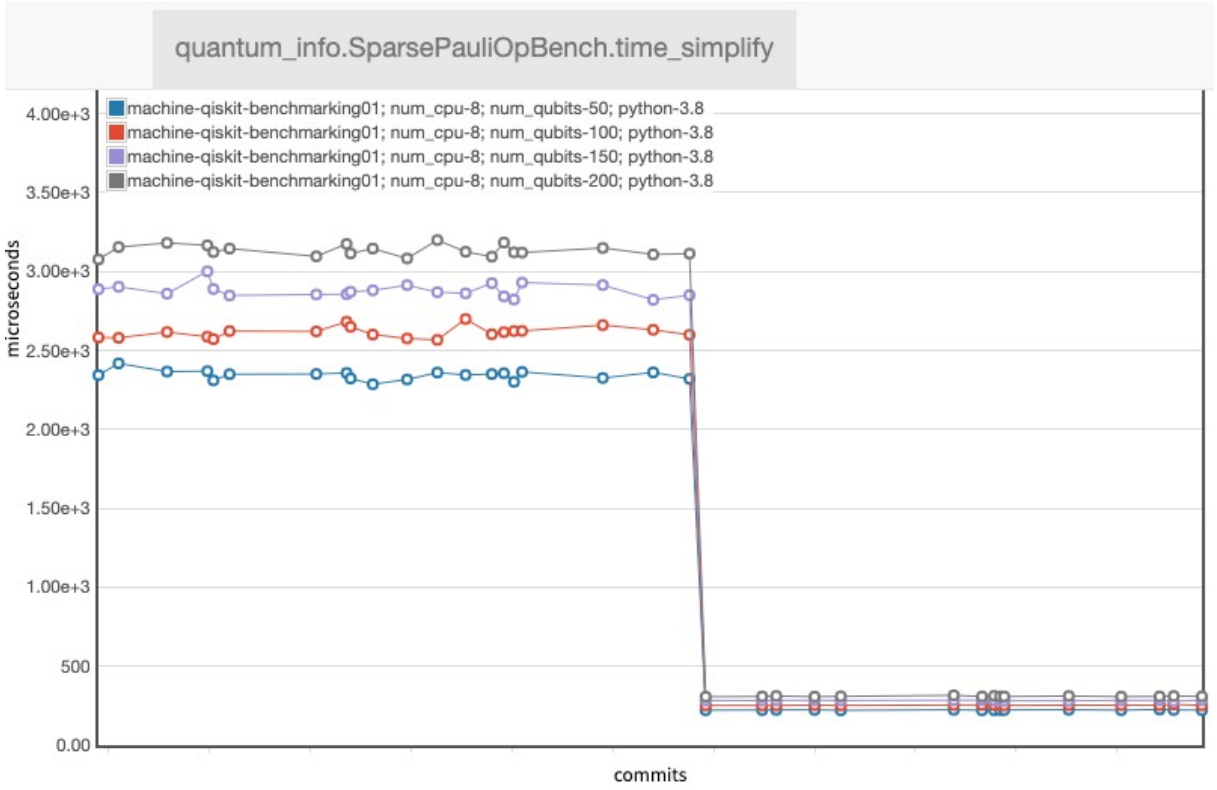
| Nature branch | Terra branch | Time |
|---|---|---|
| main | main | 61.88646222 sec |
| this PR | main | 53.311054609 sec |
| main | stable | 12.960038270999998 sec |
| this PR | stable | 10.459337683 sec |
| main | Qiskit/qiskit-terra#7122 | 12.132281858 sec |
| this PR | Qiskit/qiskit-terra#7122 | 4.158444311 sec |
| this PR | 7122 + 7126 + 7138 | 1.7728687539999999 sec |

https://github.com/Qiskit/qiskit-nature/pull/397

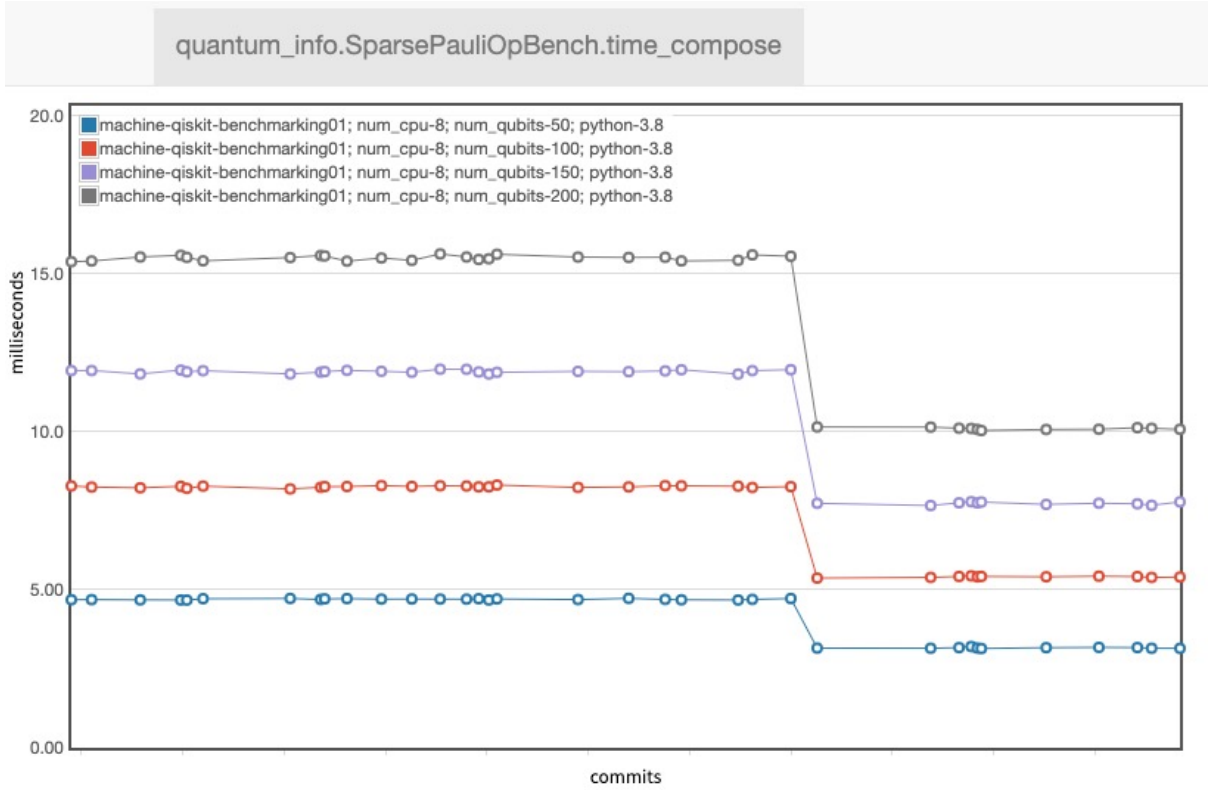
# Airspeed Velocity of Qiskit Terra

- https://qiskit.github.io/qiskit/

10x faster




35% faster



# Examples of vectorization

qiskit/quantum_info/operators/symplectic/sparse_pauli_op.py

```
@@ -341,28 +341,24 @@ def simplify(self, atol=None, rtol=None):
            if rtol is None:
                rtol = self.rtol

-        array = np.column_stack((self.paulis.x, self.paulis.z))
-        flatten_paulis, indexes = np.unique(array, return_inverse=True, axis=0)
-        coeffs = np.zeros(self.size, dtype=complex)
-        for i, val in zip(indexes, self.coeffs):
-            coeffs[i] += val
+        # Pack bool vectors into np.uint8 vectors by np.packbits
+        array = np.packbits(self.paulis.x, axis=1) * 256 + np.packbits(self.paulis.z, axis=1)
+        _, indexes, inverses = np.unique(array, return_index=True, return_inverse=True, axis=0)
+        coeffs = np.zeros(indexes.shape[0], dtype=complex)
+        np.add.at(coeffs, inverses, self.coeffs)
```

qiskit/quantum_info/operators/symplectic/pauli.py

```
@@ -601,20 +601,12 @@ def _from_label(label):
        phase = 0 if not coeff else _phase_from_label(coeff)

        # Convert to Symplectic representation
-        num_qubits = len(pauli)
-        base_z = np.zeros((1, num_qubits), dtype=bool)
-        base_x = np.zeros((1, num_qubits), dtype=bool)
-        base_phase = np.array([phase], dtype=int)
-        for i, char in enumerate(pauli):
-            if char == "X":
-                base_x[0, num_qubits - 1 - i] = True
-            elif char == "Z":
-                base_z[0, num_qubits - 1 - i] = True
-            elif char == "Y":
-                base_x[0, num_qubits - 1 - i] = True
-                base_z[0, num_qubits - 1 - i] = True
-                base_phase += 1
-        return base_z, base_x, base_phase % 4
+        pauli_bytes = np.frombuffer(pauli.encode("ascii"), dtype=np.uint8)[::-1]
+        ys = pauli_bytes == ord("Y")
+        base_x = np.logical_or(pauli_bytes == ord("X"), ys).reshape(1, -1)
+        base_z = np.logical_or(pauli_bytes == ord("Z"), ys).reshape(1, -1)
+        base_phase = np.array([(phase + np.count_nonzero(ys)) % 4], dtype=int)
+        return base_z, base_x, base_phase
```

```
@@ -216,34 +217,41 @@ def compose(self, other, qargs=None, front=False):
        # Validate composition dimensions and qargs match
        self._op_shape.compose(other._op_shape, qargs, front)

-        x1 = np.reshape(
-            np.stack(other.size * [self.paulis.x], axis=1),
-            (self.size * other.size, self.num_qubits),
-        )
-        z1 = np.reshape(
-            np.stack(other.size * [self.paulis.z], axis=1),
-            (self.size * other.size, self.num_qubits),
-        )
-        p1 = np.reshape(
-            np.stack(other.size * [self.paulis.phase], axis=1),
-            self.size * other.size,
-        )
-        paulis1 = PauliList.from_symplectic(z1, x1, p1)
-        x2 = np.reshape(
-            np.stack(self.size * [other.paulis.x]), (self.size * other.size, other.num_qubits)
-        )
-        z2 = np.reshape(
-            np.stack(self.size * [other.paulis.z]), (self.size * other.size, other.num_qubits)
-        )
-        p2 = np.reshape(
-            np.stack(self.size * [other.paulis.phase]),
-            self.size * other.size,
-        )
-        paulis2 = PauliList.from_symplectic(z2, x2, p2)
+        if qargs is not None:
+            x1, z1 = self.paulis.x[:, qargs], self.paulis.z[:, qargs]
+        else:
+            x1, z1 = self.paulis.x, self.paulis.z
+        x2, z2 = other.paulis.x, other.paulis.z
+        num_qubits = other.num_qubits

+        # This method is the outer version of `BasePauli.compose`.
+        # `x1` and `z1` have shape `(self.size, num_qubits)`.
+        # `x2` and `z2` have shape `(other.size, num_qubits)`.
+        # `x1[:, no.newaxis]` results in shape `(self.size, 1, num_qubits)`.
+        # `ar = ufunc(x1[:, np.newaxis], x2)` will be in shape `(self.size, other.size, num_qubits)`.
+        # So, `ar.reshape((-1, num_qubits))` will be in shape `(self.size * other.size, num_qubits)`.
+        # Ref: https://numpy.org/doc/stable/user/theory.broadcasting.html

+        phase = np.add.outer(self.paulis._phase, other.paulis._phase).reshape(-1)
+        if front:
+            q = np.logical_and(x1[:, np.newaxis], z2).reshape((-1, num_qubits))
+        else:
+            q = np.logical_and(z1[:, np.newaxis], x2).reshape((-1, num_qubits))
+        phase = np.mod(phase + 2 * np.sum(q, axis=1), 4)

-        pauli_list = paulis1.compose(paulis2, qargs, front)
-        coeffs = np.kron(self.coeffs, other.coeffs)
+        x3 = np.logical_xor(x1[:, np.newaxis], x2).reshape((-1, num_qubits))
+        z3 = np.logical_xor(z1[:, np.newaxis], z2).reshape((-1, num_qubits))
```

# Overhead of SparsePauliOp._add

- {SparsePauliOp, PauliList}._add is realized by stacking z, x, and phase arrays of self and other
  - Stacking arrays includes the overhead of copy
- Applying the built-in `sum` to a list of SparsePauliOp
  - Calls `_add` `len(list)` times
  - Adds a special `sum` for Qiskit nature to avoid stacking large arrays (qiskit-nature#397)
- Specialized method `SparsePauliOp.sum(list[SparsePauliOp])` would be beneficial
  - Directly stacks arrays of all operators once
  - Will make a PR for discussion

```
151     -         return PauliSumOp(sum(ret_op_list, zero_op)).reduce()
146     +         def _sum(ops: List[SparsePauliOp]) -> SparsePauliOp:
147     +             # This is equivalent to `sum`, but this reduces the overhead of `SparsePauliOp._add`,
148     +             # i.e., stack of arrays.
149     +             while len(ops) > 1:
150     +                 ops.append(ops[0] + ops[1])
151     +                 ops = ops[2:]
152     +             return ops[0]
153     +
154     +         return PauliSumOp(_sum(ret_op_list).simplify())
```

https://github.com/Qiskit/qiskit-nature/pull/397