Transpilation (Opt. Level 2) fails with registers larger than 10 bits `ValueError: too many subscripts in einsum` #XXXX

Edit New issue

Open **ANONYMOUS** opened this issue 22 days ago · 1 comment

ANONYMOUS commented 22 days ago

Environment

- **Qiskit Terra version**: 0.19.1
- **Python version**: 3.8
- **Operating system**: Ubuntu 18.04.6 LTS

What is happening?

When I transpile this circuit with optimization level 2, it strangely fails if the register is larger than 10 bits, but with 10 or less bits it works as expected.

How can we reproduce the issue?

Run this script:

```
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
qr = QuantumRegister(11, name='qr')
cr = ClassicalRegister(11, name='cr')
qc = QuantumCircuit(qr, cr, name='qc')
subcircuit = QuantumCircuit(qr, cr, name='subcircuit')
subcircuit.x(3)
qc.append(subcircuit, qargs=qr, cargs=cr)
qc.x(3)
qc.draw(fold=-1)
```

Output:

```
        ┌───────────────┐
  qr_0: ┤0              ├─────
        │               │
  qr_1: ┤1              ├─────
        │               │
  qr_2: ┤2              ├─────
        │               │┌───┐
  qr_3: ┤3              ├┤ X ├
        │               │└───┘
  qr_4: ┤4              ├─────
        │               │
  qr_5: ┤5              ├─────
        │               │
  qr_6: ┤6              ├─────
        │               │
  qr_7: ┤7              ├─────
        │               │
  qr_8: ┤8              ├─────
        │               │
  qr_9: ┤9              ├─────
        │               │
 qr_10: ┤10             ├─────
        │   subcircuit  │
  cr_0: ╡0              ╞═════
        │               │
  cr_1: ╡1              ╞═════
        │               │
  cr_2: ╡2              ╞═════
        │               │
  cr_3: ╡3              ╞═════
        │               │
  cr_4: ╡4              ╞═════
        │               │
  cr_5: ╡5              ╞═════
        │               │
  cr_6: ╡6              ╞═════
        │               │
  cr_7: ╡7              ╞═════
        │               │
  cr_8: ╡8              ╞═════
        │               │
  cr_9: ╡9              ╞═════
        │               │
 cr_10: ╡10             ╞═════
```

Assignees

No one assigned

Labels

bug

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

Notifications Customize

Unsubscribe

You're receiving notifications because you authored the thread.

2 participants

Then transpile it (with OPTIMIZATION LEVEL 2):

```
from qiskit import transpile
qc = transpile(qc, optimization_level=2)
```

Output

```
...

qiskit/quantum_info/operators/operator.py in _einsum_matmul(cls, tensor, mat, indices, shift, right_mul)
    449         else:
    450             indices_mat = mat_free + mat_contract
--> 451         return np.einsum(tensor, indices_tensor, mat, indices_mat)
    452 
    453     @classmethod

<__array_function__ internals> in einsum(*args, **kwargs)

numpy/core/einsumfunc.py in einsum(out, optimize, *operands, **kwargs)
   1359         if specified_out:
   1360             kwargs['out'] = out
-> 1361         return c_einsum(*operands, **kwargs)
   1362 
   1363     # Check the kwargs to avoid a more cryptic error later, without having to

ValueError: too many subscripts in einsum
```

To rule out possible problems with the configuration of Numpy, I tried to run the same script in a brand new Colab environment with the same Qiskit version and it still fails with the same error.

What should happen?

The transpilation should terminate without errors, similarly to what happens when using 10 or less bits.

Any suggestions?

Thanks to the stacktrace and the interactive debugger, the bug happens in the `CommutationAnalysis(AnalysisPass)`.
In particular you should step in the `_commute` function below, when the two arguments are:

```
qiskit/transpiler/passes/optimization/commutation_analysis.py(77)run()
     75                     does_commute = False
     76                     try:
3--> 77                         does_commute = _commute(current_gate, prev_gate, self.cache)
     78                     except TranspilerError:
     79                         pass
```

In particular, to reach the problematic point in code, you should step in the `_commute` function below, when the two arguments are:

```
ipdb> print(current_gate.op); print(prev_gate.op);
Instruction(name='x', num_qubits=1, num_clbits=0, params=[])
Instruction(name='subcircuit', num_qubits=11, num_clbits=11, params=[])
```

Then the problem should be in `quantum_info/operators/operator.py in _einsum_matmul` as suggested by the last part of the stack trace.

It seems very strange, I would be very interested in getting others' opinions about this...

ANONYMOUS added the bug label 22 days ago

QISKIT DEV commented 21 days ago • edited

Contributor

This can only happen while you don't have a coupling map set, either directly or by setting a `backend` argument. This is because if a coupling map is set, we decompose large operators down to at most 2-qubit operators before trying anything else. The immediate workaround is just to set a backend, then this will work.

It happens because we use Numpy's einsum to do generalised matrix-matrix multiplication on higher-order tensors during the commutation analysis pass, and it shakes out that once your operators have 11 qubits in them, we would use 33 indices to represent the tensor-reduction summation we want. Numpy has an upper limit on the number of indices it can use in an array (or any description of an array iterator), which is `numpy.MAXDIM` and is 32 in my installation (and presumably yours).

This would mean that this pass will always fail for commutation analysis of two 11+ qubit operators. We could possibly handle that by just assuming that if two operators are that large and have overlapping qubit inputs, then they don't commute - the cost of the matrix multiplication is so high it hardly matters anyway, and 11+ qubit operators are very rare. In this particular case, where there's one very large operator and one very small one, there's *possibly* a little trick we could play in `CommutationAnalysis` to ensure that the larger operator gets passed in the correct slot to trigger a smaller einsum - it's quite possible the pass is currently using `Operator.compose` the wrong way round for that. But in this particular implementation of commutation analysis, the best solution is probably just to give up and say "they probably don't commute" if the operators are too large.

1

Write Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Close issue Comment

Remember, contributions to this repository should follow its contributing guidelines and code of conduct.

Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About