TorchQuantum Unitary Fund Application

Hanrui Wang, Advisor: Prof. Song Han
MIT

TorchQuantum

- Classical simulation of quantum circuit with PyTorch interface
  - Automatic **gradient computation**
  - **GPU**-accelerate, **batch** mode
  - **Dynamic** computation graph
  - **hybrid** classical and quantum computation
  - **Gate** level and **pulse** level
  - **Converters** to other frameworks such as IBM Qiskit

Implementation

- Statevector simulator

```
_state = torch.zeros(2**self.n_wires, dtype=C_DTYPE)
_state[0] = 1 + 0j  # type: ignore
_state = torch.reshape(_state, [2] * self.n_wires).to(self.device)
self.register_buffer("state", _state)

repeat_times = [bsz] + [1] * len(self.state.shape)  # type: ignore
self._states = self.state.repeat(*repeat_times)  # type: ignore
self.register_buffer("states", self._states)
```

```
"cnot": torch.tensor(
    [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 0, 1], [0, 0, 1, 0]], dtype=C_DTYPE
),
```

```
mat = mat.type(C_DTYPE).to(state.device)

devices_dims = [w + 1 for w in device_wires]
permute_to = list(range(state.dim()))
for d in sorted(devices_dims, reverse=True):
    del permute_to[d]
permute_to = permute_to[:1] + devices_dims + permute_to[1:]
permute_back = list(np.argsort(permute_to))
original_shape = state.shape
permuted = state.permute(permute_to).reshape([original_shape[0], mat.shape[-1], -1])

if len(mat.shape) > 2:
    # both matrix and state are in batch mode
    new_state = mat.bmm(permuted)
else:
    # matrix no batch, state in batch mode
    bsz = permuted.shape[0]
    expand_shape = [bsz] + list(mat.shape)
    new_state = mat.expand(expand_shape).bmm(permuted)

new_state = new_state.view(original_shape).permute(permute_back)
```

Development Plan

- Add **density matrix** simulation
- Support more comprehensive **noise models** that use Kraus operators, instead of simple Paulis.
- More comprehensive **pulse simulation** by solving Schrödinger/Master equation on GPU to enable faster pulse optimization and QOC.
- Integrate **cuQuantum** to TQ.

Thank you!