量子计算
—编程篇

Quantum Computer

网址：www.qubits.top

作者：Calvin Tang

邮箱：179209347@qq.com

介绍

本开发教程基于本源量子的Qpanda框架的python版 – **PyQPanda** 编写。

- 一种功能齐全，运行高效的量子软件开发工具包
- QPanda 2是由本源量子开发的开源量子计算框架，它可以用于构建、运行和优化量子算法。
- QPanda 2作为本源量子计算系列软件的基础库，为OriginIR、Qurator、量子计算服务提供核心部件。

QPanda使用文档：
https://pyqpanda-toturial.readthedocs.io/zh/latest/index.html

Github & Gitee 代码地址：
https://github.com/mymagicpower/quantum/tree/main/quantum_qpanda/tools
https://gitee.com/mymagicpower/quantum/tree/main/quantum_qpanda/tools

量子线路查询替换

在量子计算中，存在一些量子逻辑门或量子线路是可以相互替代的，比如如下替换过程：

H(0)->CNOT(1,0)->H(0) 可以替换为 CZ(1,0)。

在量子程序中，可能存在多个相同结构的子量子线路或多个相同的量子逻辑门，查询替换量子程序中指定结构的量子线路的功能就是找这些相同结构的子量子线路并把它们替换成目标量子线路。

提供了统一的量子线路优化口：circuit_optimizer ，该接口可实现多种量子线路的查询替换，对应的接口参数： 参数1：QProg 待优化的原始量子程序 参数2：vector 子线路查询替换队列，每个队列元素包含目标搜索线路和对应的替换线路。

https://pyqpanda-toturial.readthedocs.io/zh/latest/GraphMatch.html

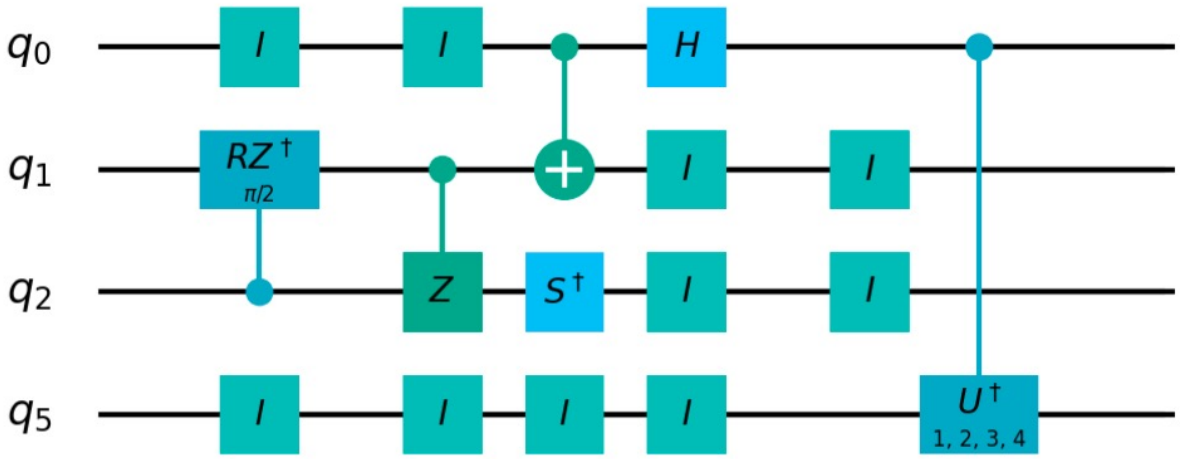
0_Circuit_Optimizer.py

```
...
    prog << H(q[0])\
         << H(q[2])\
         << H(q[3])\
         << CNOT(q[1], q[0])\
         << H(q[0])\
         << CNOT(q[1], q[2])\
         << H(q[2])\
         << CNOT(q[2], q[3])\
         << H(q[3])
    # 构建查询线路
         query_cir = QCircuit()
         query_cir << H(q[0])<< CNOT(q[1], q[0])<< H(q[0])
    # 构建替换线路
         replace_cir = QCircuit()
    replace_cir << CZ(q[1], q[0])
    print("查询替换前：")
    print(convert_qprog_to_originir(prog,machine))
    # 搜索量子程序中的查询线路，并用替换线路替代
     update_prog = circuit_optimizer(prog, [[query_cir, replace_cir]])
    print("查询替换后：")
    print(convert_qprog_to_originir(update_prog,machine))
```

用I门填充QProg

接口 fill_qprog_by_I 实现用I门填充QProg(量子程序)的功能。

示例程序演示了 fill_qprog_by_I 接口的使用方法，只需要传入一个QProg类型的参数即可，该接口返回一个填充后的新QProg，输入QProg保持不变。
以上示例程序的字符画展示输出结果如下：



1_Fill_Qprog_By_I.py

```
...
   # 构建量子程序
        prog = pq.QCircuit()
    prog << pq.CU(1, 2, 3, 4, q[0], q[5]) << pq.H(q[0]) << pq.S(q[2]) <<
pq.CNOT(q[0], q[1]) << pq.CZ(q[1], q[2]) << pq.CR(q[2], q[1], math.pi/2)
    prog.set_dagger(True)
   # 输出原量子程序
  print('source prog:')
    draw_qprog(prog, 'text',console_encode_type='gbk')
   # 量子程序填充 I 门
  prog = pq.fill_qprog_by_I(prog)
   # 输出填充 I 门的量子程序
  print('The prog after fill_qprog_by_I:')
    draw_qprog(prog, 'text',console_encode_type='gbk')
    draw_qprog(prog, 'pic', filename='./test_cir_draw.png')
if __name__=="__main__":
    init_machine = InitQMachine(16, 16)
    qlist = init_machine.m_qlist
    clist = init_machine.m_clist
    machine = init_machine.m_machine
    fill_I(qlist, clist)
```

酉矩阵分解

问题

目前，量子计算的算法通常用量子线路表示，量子线路包括量子逻辑门操作。 通常，连续的一段量子线路通常包含几十上百个甚至成千上万个量子逻辑门操作，而量子逻辑门数量或单个量子逻辑门操作的量子比特数越多，计算过程越为复杂，导致量子线路的模拟效率较低，且对硬件资源的占用较多。

算法目标

对于上述问题，有必要对量子线路进行一种等价转换，需要减少量子线路中逻辑门的数量， 同时在此基础上，需要确保转换前后整个量子线路对应的酉矩阵完全相同。

算法概述

本文介绍的算法是将一个N阶酉矩阵，分解成不超过r = N(N−1)/2个带有少量控制的单量子逻辑门序列，其中$N=2^n$，分解的产物满足如下等式关系。

$$U_r U_{r-1} \cdots U_3 U_2 U_1 U = I_N$$

从而可以得到原矩阵U的分解结果表示：

$$U = U^\dagger_1 U^\dagger_2 U^\dagger_3 \cdots U^\dagger_{r-1} U^\dagger_r$$

Calvin, QQ: 179209347 Mail: 179209347@qq.com

酉矩阵分解

pyqpanda中设计了 matrix_decompose 接口用于进行酉矩阵分解，该接口需要两个参数， 第一个是使用到的所有量子比特，第二个是待分解的酉矩阵，该函数的输出是转换后的量子线路。

```
source matrix :
[(0.6477054522122977+0.11954177678702191j), (-0.16162176706189357-0.4020495632468249j), (-0.19991615329121998-0
 .3764618308248643j), (-0.2599957197928922-0.3593524887300786j), (-0.16162176706189363-0.40204956324682495j), (0
 .7303014482204584-0.4215172444390785j), (-0.15199187936216693+0.09733585496768032j), (-0.22248203136345918-0
 .13836005976607446j), (-0.19991615329122003-0.3764618308248644j), (-0.15199187936216688+0.09733585496768032j), (0
 .6826630277354306-0.37517063774206166j), (-0.3078966462928956-0.2900897445133085j), (-0.2599957197928923-0
 .3593524887300787j), (-0.22248203136345912-0.13836005976607446j), (-0.30789664629289554-0.2900897445133085j), (0
 .6640994547408099-0.338593803336005j)]
the decomposed matrix :
[(0.6477054522122979+0.11954177678702192j), (-0.16162176706189357-0.402049563246825j), (-0.19991615329122014-0
 .3764618308248645j), (-0.2599957197928924-0.3593524887300788j), (-0.16162176706189368-0.40204956324682506j), (0
 .7303014482204584-0.4215172444390785j), (-0.15199187936216693+0.09733585496768031j), (-0.22248203136345918-0
 .13836005976607446j), (-0.19991615329122003-0.3764618308248644j), (-0.151991879362167+0.097335854967680042j), (0
 .6826630277354307-0.3751706377420617j), (-0.30789664629289576-0.2900897445133086j), (-0.2599957197928924-0
 .359352488730078 75j), (-0.22248203136345918-0.13836005976607443j), (-0.3078966462928958-0.290089744513308 66j), (0
 .6640994547408103-0.3385938033360052j)]
matrix decompose ok !
```

2_Matrix_Decompose.py

```
...
if __name__=="__main__":
    machine = pq.init_quantum_machine(pq.QMachineType.CPU)
    q = machine.qAlloc_many(2)
    c = machine.cAlloc_many(2)
    source_matrix = ....
    print("source matrix : ")
    print(source_matrix)
    out_cir = pq.matrix_decompose(q, source_matrix)
    circuit_matrix = pq.get_matrix(out_cir)

    print("the decomposed matrix : ")
    print(circuit_matrix)

    source_matrix = np.round(np.array(source_matrix),3)
    circuit_matrix = np.round(np.array(circuit_matrix),3)
    if np.all(source_matrix == circuit_matrix):
        print('matrix decompose ok !')
    else:
        print('matrix decompose false !')
```

https://pyqpanda-toturial.readthedocs.io/zh/latest/MatrixDecompostion.html

Thank You