ProjectQ로 설계한 양자 Carry-Sum Adder

서창배, 이재홍, 조성민, 서승현
한양대학교 전자공학부
e-mail:cbseook@hanyang.ac.kr

Design Modified Quantum Carry-Sum Adder through ProjectQ

Chang-Bae Seo, Jae-Hong Lee, Seong-Min Cho, Seugn-Hyun Seo
Electronic Engineering, Hanyang University

요 약

최근까지도 양자 하드웨어의 개발은 꾸준히 이루어졌지만, 개발 수준이 양자 소프트웨어의 연구에 사용하기에 부족한 정도이다. 따라서 양자 하드웨어 없이도 양자 소프트웨어의 연구를 진행하기 위해서는 양자 시뮬레이터와 컴파일러가 필요해졌다. 이에 다양한 양자 시뮬레이터와 컴파일러가 제공되었으며 양자 시뮬레이터와 컴파일러가 하나의 소프트웨어 프레임워크를 이루고 있는 풀-스택 라이브러리 역시 다양하게 제공되고 있다. ProjectQ는 풀-스택 라이브러리 중 하나로써 Python을 기반으로 하여 무료로 사용 가능할 뿐만 아니라 문법이 쉬워 접근성이 높다는 장점이 있고, 컴파일러에 시뮬레이터와 에뮬레이터의 효율적인 적용이 가능하여 새로운 양자 알고리즘 개발의 가속화나 양자 회로의 시각화 등이 가능하다. 따라서 본 논문은 ProjectQ의 내부 구성과 기능을 구체적으로 설명한 후, 기존의 Carry-Sum adder를 응용한 새로운 양자 회로를 직접 구현해보았다.

1. 서론

최근 Intel은 양자 프로세서를 탑재한 72 큐비트 칩을 발표하였으며, IBM은 2019년 10월 안에 단일규모로는 최대인 53 큐비트 범용 양자시스템의 클라우드 서비스를 제공할 예정이다. 앞선 경우와 같이 양자 하드웨어의 개발은 꾸준히 이루어지고 있었으나, 발전 수준은 아직 보급하여 사용할 수준에 이르지는 못한다. 이는 곧 양자 컴퓨터를 통해 양자 알고리즘이나 회로를 구현해보고, 최적화하는 소프트웨어의 연구에 사용하기에는 현실적으로 무리인 수준이다. 그렇기에 현시점에서 양자 소프트웨어의 연구가 이루어지기 위해서는 양자 시뮬레이터와 컴파일러의 사용이 필요한 상황이다. 본 논문에서는 ProjectQ라는 python 기반의 풀-스택 라이브러리를 소개한다. 풀-스택 라이브러리는 양자 컴파일과 시뮬레이션이 하나로 구성된 소프트웨어 프레임워크를 의미하며 양자 알고리즘의 구현이나 최적화 등이 가능하다. ProjectQ 외의 풀-스택 라이브러리로는 프로그래밍 언어별로 C++에서는 XACC, python에서는 IBM의 Qiskit이나 D-wave의 XACC 또는 Rigetti의 Forest, JavaScipt에서는 Qiskit-Js, Microsoft의 별로 양자 프로그래밍 언어를 활용하는 Q# 등이 있다.

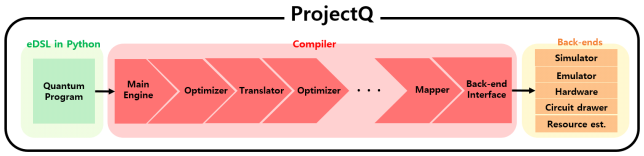
ProjectQ는 양자 풀-스택 라이브러리 중 하드웨어에 구애받지 않고 컴파일러에 효율적으로 시뮬레이터와 에뮬레이터를 적용할 수 있어 새로운 양자 알고리즘을 개발을 가속화 할 수 있고, IBM의 양자컴퓨터에 개방형 클라우드 서비스를 통해 접속할 수 있어 실제 양자컴퓨터로 돌려볼 수 있어 양자 컴퓨터에서의 양자 알고리즘 구현과 시뮬레이션 및 회로의 시각화 기능을 제공하고 있다[5]. 따라서 양자 알고리즘 설계 시 ProjectQ를 이용하면 설계한 양자 회로를 시각화하여 볼 수 있으며, 게이트 통과에 따른 큐비트의 상태 관측 후 큐비트의 붕괴 결과의 확인이 가능하여 디버깅에 도움을 줄 수 있다. 이에 본 논문에서는 양자 게이트를 이용한 큐비트의 덧셈 회로인 Carry-Sum adder를 ProjectQ로 설계하는 구체적인 방법을 제시하고, 시뮬레이션한 결과를 확인해보았다.

이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2019-0-00033, 미래컴퓨팅 환경에 대비한 계산 복잡도 기반 암호 안전성 검증 기술개발).
본 연구는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No.2018R1A2B6006903).

2. ProjectQ의 구성

ProjectQ는 (그림 1)과 같이 'eDSL in Python', 'Compiler', 'Back-ends'의 3가지로 구성되어있다.

2.1 eDSL in Python



(그림 1) ProjectQ 구성

eDSL in Python은 양자 회로를 설계하기 위해 Python 언어로 프로그램 코드를 작성하는 방법이라고 할 수 있으며, 기본적으로 사용할 기능이나 함수들을 import하거나 gate들을 사용하는 방법에 대하여 알아야 한다.

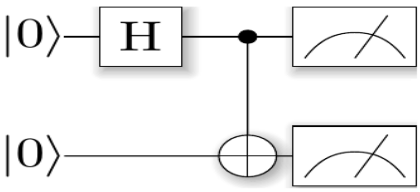
2.2 Compiler

컴파일러는 설계한 양자 알고리즘에 맞춰 작성한 python 코드에 양자 특성을 반영시켜 회로를 그렸을 때 정상적인 동작을 수행하는지 확인하기 위해 사용한다. ProjectQ의 컴파일러는 (그림 1)에서 확인할 수 있듯 모듈화되었고, 체인 연결 방식을 통해 사용자의 커스터마이징을 빠른 속도로 반영하는 front-end 기능을 수행한다. 게다가 컴파일하는 동안 지속적인 최적화가 이루어지게 설계되었고, 사용할 Back-end에 맞게 Interface가 가능하다.

2.3 Back-ends

ProjectQ는 다음 5가지의 back-ends를 지원한다.

- Hardware back-end: IBM의 Quantum Experience chip에서의 시뮬레이션을 제공한다.
- Simulation of quantum circuits: 양자 회로의 개별 게이트에 대한 시뮬레이션을 제공한다.
- Emulation of quantum circuits: high-level shortcuts의 사용을 통한 양자 회로 작용의 에뮬레이션을 제공한다.
- Resource estimation: 양자 회로 구현에 소요되는 자원 추정을 제공한다.
- Circuit drawing back-end: 양자 회로 다이어그램의 드로잉을 제공하며 (그림 2)와 같은 회로를 직접 그려 볼 수 있다.



(그림 2) tex 파일을 통해 pdf로 그린 bell pair circuit

3. 양자 알고리즘 설계에 필요한 게이트 및 기능

3.1 양자 회로 및 게이트 사용

<코드 1> ProjectQ 예제 code

```
from projectq import MainEngine
from projectq.ops import H, Measure

# create a main compiler engine
eng= MainEngine()
# allocate one qubit
x1=eng.allocate_qubit()
# put it in superposition
H | x1

# measure & flush
Measure | x1
eng.flush()
# print the result:
print("Measured: {}".format(int(x1)))
```

<코드 1>은 컴파일러는 MainEngine을 사용하고, 게이트는 ops(operations)에 포함된 H(=Hadamard gate)와 Measure를 사용한다는 것을 알 수 있다. <코드 1>을 통해 양자비트를 gate에 통과시키는 방법을 알 수 있으며, 그 방법은 아래와 같다.

'gate | (사용하거나 결과를 저장할 qubit들)'

따라서 H | x1의 의미는 x1에 할당받은 qubit을 중첩 상태를 만들어주는 것이고, Measure | x1은 x1에 할당되는 qubit을 관측하는 것이다.

※모듈화

ProjectQ에서는 <코드 2>와 같이 회로를 함수의 형태로 모듈화하여 사용하는 것을 권장한다. 이는 모듈화한 코드가 최적화나 게이트 합성 등의 부분에서 더 유리하고, 컴파일이 개선된 경우 더 쉽게 적용할 수 있기 때문이다.

<코드 2> 모듈화

```
from projectq.ops import H, CNOT

def create_bell_pair(eng):
    b1 = eng.allocate_qubit()
    b2 = eng.allocate_qubit()
    H | b1
    CNOT | (b1, b2)
    return b1, b2
```

3.2 ProjectQ의 최적화 함수

본 절에서는 복잡한 게이트 사용을 최적화가 가능하고, 빠른 프로토타이핑이나 확장성을 제공해줄 수 있는 meta 함수들의 기능을 소개한다.

- Loop
  설계한 양자 회로가 특정 양자 함수나 회로들을 반복 사용이 필요한 경우 활용한다.
  'with Loop(eng, num):'
  위와 같은 형식으로 사용하며, num에 해당하는 값만큼 반복한다.
- Compute & Uncompute
  Compute는 특정한 기능을 수행하게 설계 할 수 있고, Uncompute는 설계한 Compute의 동작을 반대로 수행하는 기능이다.
  'with Compute(eng):

All(H) | x
All(X) | x
⋮
Uncompute(eng)'

Compute가 x에 해당하는 큐비트들에 H gate를 적용하고 X gate를 적용하는 식으로 진행된다면 Uncompute는 반대되는 순서로 X gate를 적용하고 H gate를 적용하는 순서로 적용된다.

- Dagger

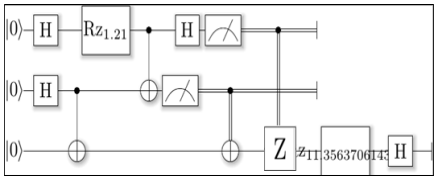
Dagger는 양자 회로에서 전치 켤레를 수행해주어야 하는 경우 활용한다.

'with Dagger(eng):
state_creation_function(eng, b2)'

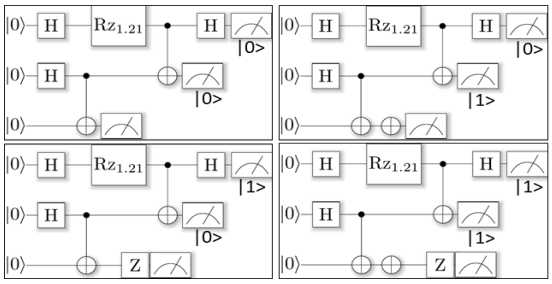
위의 경우에는 state_creatin_function이라는 함수에서 수행한 기능에 대해 전치 켤레시킨 기능을 수행한다.

- Control

특정한 상황에만 어떠한 동작을 수행해야 하는 경우 활용한다. Control 기능은 프로그램 코드의 조건문과 비슷한 역할을 수행하는 것 같지만, Control기능을 사용한 (그림 3)와 조건문을 사용한 (그림 4)을 비교하면 그 차이를 알 수 있다.



(그림 3) control 기능으로 그린 회로



(그림 4) 조건문으로 그린 회로

'with Control(eng, x[0:-1]):
Z | x[-1]'

위의 경우에는 x[0:-1]에 해당하는 모든 큐비트가 1일 때에만 Z | x[-1]이 적용되는 역할을 수행한다.

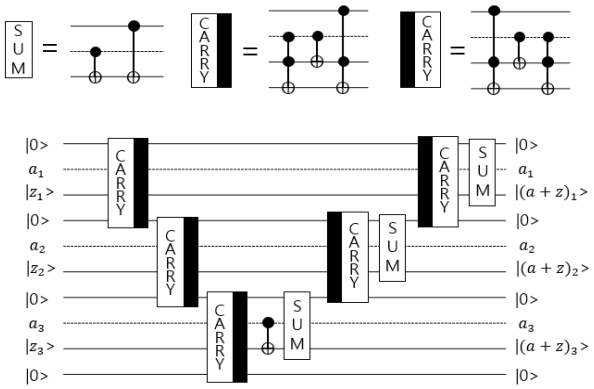
추가로 ControlledGate를 수행하면 위의 간단한 예시의 역할을 똑같이 수행할 수 있다.

'ControlledGate(Z,3) | (x[0], x[1], x[::-1], x[-1])'

그러나 ControlledGate와 meta 기능인 Control을 사용할 때의 차이를 비교해보면 ControlledGate의 경우에는 control한 결과가 한 비트에 적용될 뿐이지만, meta 기능의 Control을 이용하면 다양한 gate들로 구성된 회로를 control할 수 있다는 차이가 있다.

4. 양자 회로 설계(carry-sum adder)

[steading]에서는 양자 게이트들을 사용하여 기본 덧셈 연산을 구현한 Carry-Sum adder를 소개하고 설명하고 있으며, 회로는 (그림 5)에서 확인할 수 있다. 그러나 [6]에서 제시하는 Carry-Sum adder는 정해진 값만을 더하는 adder이기 때문에 $a_1 \sim a_3$는 큐비트가 아니다.



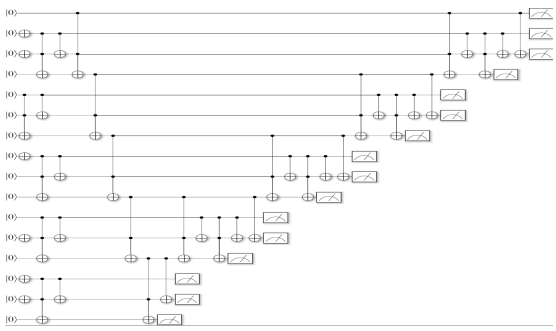
(그림 5) 기본 Carry-Sum adder

본 장에서는 큐비트끼리의 연산을 고려하여 a의 위치에 큐비트를 사용하고, 덧셈 연산을 수행할 두 수와 그 수의 비트 길이를 입력값으로 받아 덧셈 결과를 출력하는 응용된 버전의 Carry-Sum adder를 ProjectQ를 통해 설계하고 연산 결과를 확인해보았다. 또한, ProjectQ에서 제공하는 back-end 기능 중 하나인 circuit drawing을 통해 양자 회로 다이어그램을 그려서 Carry-Sum adder의 양자 회로를 시각화하여 보았다.

<코드 3> 응용된 carry-sum adder

```
from projectq.ops import H, Measure , X, All, ControlledGate, CNOT
from projectq import MainEngine
from projectq.backends import CircuitDrawer

System_signal=1
result = 0
tens = 1
Bit = input("if you want 5 bit's adding, you should entered '5' \ninput Bits: ")
Adder = input("\n!!!TIP!!! \n ex) if you wanted add 5 bits and then you want adder's value
is 3 \n-> you should enter 00011\nThe adder's value?  : ")
Num = input("\nWhat's the number you want to add to the adder |{}>\nPut it in line with
the Bits you entered\nNumber : ".format(Adder))

Num_length = len(Num)
Adder_length = len(Adder)

if Num_length == Adder_length  :
    for i in range(Num_length):
        if ((Num[i] != "0") and (Num[i] != "1")) :
            print("\nyou didn't entered Number's 'bit' (consisting of 0 or 1)")
            System_signal=0
            break
        elif ((Adder[i] != "0") and (Adder[i] != "1")) :
            print("\nyou didn't entered Adder's 'bit' (consisting of 0 or 1)")
            System_signal=0
            break
else :
    print("\nYou didn't fit the bits of the value of Adder and Number.")
    System_signal=0

drawing_circuit = CircuitDrawer()
```

```
eng=MainEngine(drawing_circuit)
#eng = MainEngine()
qureg = eng.allocate_qureg(1+Num_length*3)

for i in range(Num_length):
    if Num[i] == "1" :
        X | qureg[2+(Num_length-i-1)*3]
for i in range(Adder_length):
    if Adder[i] == "1":
        X | qureg[1+(Num_length-i-1)*3]

def Carry(eng, qureg, sequence) :
    ControlledGate(X, 2) | (qureg[sequence*3+1], qureg[sequence*3+2], qureg[sequence*3+3])
    CNOT | (qureg[sequence*3+1], qureg[sequence*3+2])
    ControlledGate(X, 2) | (qureg[sequence*3], qureg[sequence*3+2], qureg[sequence*3+3])
    return qureg
def Sum(eng, qureg, sequence) :
    CNOT | (qureg[(Num_length-sequence)*3-2], qureg[(Num_length-sequence)*3-1])
    CNOT | (qureg[(Num_length-sequence)*3-3], qureg[(Num_length-sequence)*3-1])
    return qureg
def Inverse_Carry(eng, qureg, sequence):
    ControlledGate(X, 2) | (qureg[sequence*3], qureg[sequence*3+2], qureg[sequence*3+3])
    CNOT | (qureg[sequence*3+1], qureg[sequence*3+2])
    ControlledGate(X, 2) | (qureg[sequence*3+1], qureg[sequence*3+2], qureg[sequence*3+3])
    return qureg

if System_signal == 1:
    for sequence in range(Num_length):
        qureg = Carry(eng, qureg, sequence)
    CNOT | (qureg[3*Num_length-2], qureg[3*Num_length-1])
    for sequence in range(Num_length):
        if sequence > 0 :
            Inverse_Carry(eng, qureg, Num_length-sequence-1)
        Sum(eng, qureg, sequence)
All(Measure) | qureg
eng.flush()

#print result
if System_signal == 1:
    for sequence in range(1+Num_length*3):
        print("{}".format( int(qureg[sequence]) ))

    for i in range(Num_length):
        result+=int(qureg[(i)*3+2]) * tens
        tens *= 10
    result += int(qureg[(i+1)*3]) * tens
    print("\tadder \t\t+ \t\tnumber \t\t=>\tresult")
    print("\t|{}> \t\t+ \t\t|{}> \t\t=>\t|{}>".format(Adder, Num, result))
    print(drawing_circuit.get_latex())
```



(그림 6) 응용된 회로 결과

```
if you want 5 bit's adding, you should entered '5'
input Bits: 5

!!!TIP!!!
 ex) if you wanted add 5 bits and then you want adder's value is 3
-> you should enter 00011
The adder's value?  : 10101

What's the number you want to add to the adder |10101>
Put it in line with the Bits you entered
Number : 11001
0
1
0
0
0
1
0
1
1
0
0
1
0
1
0
1

        adder           +               number          =>      result
        |10101>        +               |11001>         =>      |101110>
```

(그림 7) 응용된 회로 실행 결과

6. 결론

양자 회로를 설계해볼 수 있는 소프트웨어 프레임워크는 많지만 ProjectQ는 무료로 오픈되어 있고, python을 기반으로 하여 접근성이 좋다는 장점이 있다. 또한 IBM-Q의 컴퓨터에 연결하여 실제 양자 컴퓨터에서의 연산 결과를 확인할 수도 있으며 회로의 성능 및 상태 확인에 용이하여, 더 효율적으로 양자 회로나 양자 알고리즘을 설계하기에 적합하다. 이에 본 논문에서는 ProjectQ에 대해 소개한 뒤, 실제로 이를 이용하여 양자 회로를 구현하고 시뮬레이션해 보았다. 이를 위해 ProjectQ에서 제공하고 있는 양자 게이트 및 기능의 사용 방법에 대해 서술한 뒤, 실제 ProjectQ를 이용하여 양자 게이트를 이용한 큐비트의 덧셈 회로인 Carry-Sum adder를 응용하여 새로운 방식으로 설계해보고 구현 및 시뮬레이션을 하여 결과를 확인해보았다.

참고문헌

[1] P. W. Shor, “Algorithms for quantum computation: discrete logarithms and factoring”, In SFCS ’94 Proceedings of the 35th Annual Symposium on Foundations of Computer Science, pages 124 - 134. IEEE, 1994.

[2] P. W. Shor, “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer”, SIAM Journal on Computing, 26:1484 - 1509, 1997.

[3] L. K. Grover, “A fast quantum mechanical algorithm for database search”, In Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pages 212 - 219. ACM, 1996.

[4] P. Wchwabe and B. Westerbaan, “Solving binary MQ with Grover’s Algorithm”, In: Carlet, C.; Hasan, A.; Saraswat, V.(ed.), Security, Privacy, and Advanced Cryptography Engineering: 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016. pp. 303 - 322. Berlin: Springer-Verlag (2016)

[5] D. S. Steiger, T. Haner, and M. Troyer, “ProjectQ: An open source software framework for quantum computing”, arXiv:1612.08091, 2016.

[6] S. Beauregard, G. Brassard, and J. M. Fernandez, “Quantum Arithmetic on Galois Fields”, ArXiv.org Preprint quant-ph/0301163, 2002.