New Physics: Sae Mulli,
Vol. 72, No. 12, December 2022, pp. 938∼945
http://dx.doi.org/10.3938/NPSM.72.938

Introduction to Quantum Computing with IBM Quantum Experience Using Qiskit and OpenQASM

Jong Wan Lee*
School of Nano Convergence Technology, Hallym University, Chuncheon 24252, Korea

(Received 1 October 2022 : revised 13 November 2022 : accepted 13 November 2022)

Quantum Information Software Kit (Qiskit) is an open-source SDK for working with quantum computers of IBM Quantum Experience, and it is available for everyone who is interested in programming real quantum computers. Open Quantum Assembly Language (OpenQASM) is an intermediate representation of quantum instructions. Jupyter notebook is a web-based interactive computational environment for the creation of notebook documents for Python language. In this paper, we introduce four compact versions of Jupyter notebook environment to help students start programming with IBM Quantum Experience using Qiskit and OpenQASM.

Keywords: Quantum computing, IBM Quantum Experience, Qiskit, OpenQASM, Physics education

IBM Quantum Experience의 Qiskit과 OpenQASM을 이용한 양자 계산의 소개

이종완*
한림대학교 나노융합스쿨, 춘천 24252, 대한민국

(2022년 10월 1일 받음, 2022년 11월 13일 수정본 받음, 2022년 11월 13일 게재 확정)

Qiskit(Quantum Information Software Kit)은 IBM Quantum Experience의 양자 컴퓨터 계산을 위한 오픈 소스 소프트웨어 개발 키트로 실제 양자 컴퓨터 프로그래밍에 관심이 있는 모든 사람들이 사용할 수 있다. OpenQASM(Open Quantum Assembly Language)은 양자 명령어들의 어셈블리 언어이다. Jupyter Notebook은 Python 언어용 노트북 문서를 생성하기 위한 웹 기반 대화형 컴퓨팅 환경이다. 이 논문에서는 학생들이 Qiskit 및 OpenQASM을 사용하여 IBM Quantum Experience으로 프로그래밍을 시작하는 데 도움이 될 Jupyter Notebook 환경의 4가지 축약된 버전을 소개한다.

Keywords: 양자 계산, IBM Quantum Experience, Qiskit, OpenQASM, 물리 교육

I. 서 론

1982년에 물리학자 리처드 파인만 교수는 세계는 원자로 이루어져 있고, 그것을 시뮬레이션하기 위해서는 양자 물리학에 기반을 둔 새로운 컴퓨팅 모델로 양자 계산(Quan-

*E-mail: jwlee@hallym.ac.kr

tum Computing)이 필요하다고 역설하였다 [1]. 양자 컴퓨터(Quantum Computer)의 등장을 암시한 것이다. 양자 컴퓨터의 하드웨어에 대해서는 한국물리학회의 물리학과 첨단기술에 잘 요약되어 있다 [2]. 그 내용으로는 이온 포획장치를 이용한 양자 컴퓨터, 초전도 양자컴퓨터의 현재 수준과 활용, 광학기반 양자컴퓨터를 향하여 등이 기록되어 있다. 2019년에는 Google이 양자 우월성 (Quantum Supremacy)에 관한 논문을 Nature에 발표하기에 이르렀다 [3]. 그 이후 많은 연구자들의 노력으로 양자 컴퓨터는 이제 누구나 사용할 수 있는 툴로 자리잡게 되었다. 그 중심에 IBM Quantum Experience가 있다 [4]. IBM은 세계 최초로 양자 컴퓨터를 일반인에 공개해 누구나 데스크톱이나 랩톱으로 클라우드에 접속해 양자 컴퓨터를 사용할 수 있게 하였다. IBM 외에도 Amazon Braket [5]와 Microsoft Azure Quantum [6] 등도 클라우드 서비스를 제공하고 있으며 국내에서는 양자정보연구지원센터 [7]를 통하여 무료로 클라우드 서비스를 제공 받을 수 있다.

Qiskit(Quantum Information Software Kit) [8, 9]은 오픈 소스 SDK(Software Development Kit)로 양자 실험, 프로그램, 애플리케이션을 작성하는 양자 계산 프레임워크이다. Qiskit은 양자 계산을 시뮬레이션하는 툴과 API(Application Program Interface)를 통해 IBM Quantum Experience에 접근하는 파이썬(Python) [10] 인터페이스를 제공한다. Qiskit을 이용한 다양한 사용 방법들이 소개되고 있다 [11–13]. Qiskit은 OpenQASM(Open Quantum Assembly Language)을 통해 어셈블리 언어로 양자 회로(Quantum Circuit)를 작성할 수 있다.

기존의 디지털 컴퓨터 또는 고전 컴퓨터는 정보의 최소 단위로 0과 1인 비트(Bit)를 사용하고 있지만 양자 컴퓨터에서는 양자 상태 $|0\rangle$ 과 $|1\rangle$ 그리고 이들의 중첩(Superposition) 상태

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1}$$

인 큐비트(Qubit: Quantum Bit)를 사용하고 있다. 여기서 α와 β는 복소수로 다음 관계식을 만족한다.

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2}$$

우리가 큐비트를 측정을 할 때 결과가 0일 확률이 $|\alpha|^2$이고 결과가 1일 확률이 $|\beta|^2$이다 [14].

Figure 1은 Bell 상태(state) 또는 EPR 쌍(pair)이라고 불리는 얽힘(Entanglement) 상태의 양자 회로를 나타내고 있다. H로 표시된 Hadamard 게이트와 이어진 CNOT(Controlled-Not Gate) 게이트가 양자 레지스터(Quantum Register)이면서 큐비트인 q0와 q1에 작용하고

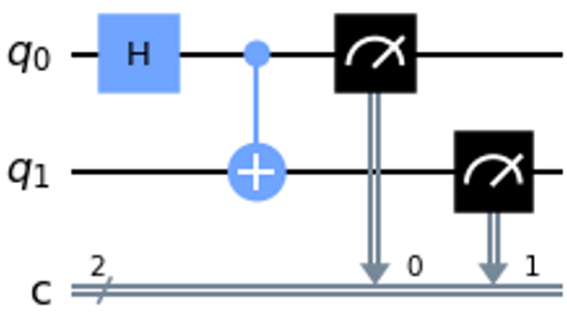


Fig. 1. (Color online) Quantum circuit diagram of the Bell state or EPR pair. Hadamard and CNOT quantum gate have been applied to the quantum register q0 and q1. The results of the quantum measurement are placed to the classical register c.

있다. q0와 q1의 초기 상태는 각각 $|0\rangle$이다. Hadamard 게이트를 q0에 작용하면 q0는 다음과 같은 중첩 상태에 놓인다.

$$\frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle \tag{3}$$

여기서 측정할 때 결과가 0일 확률이 50% ($|1/\sqrt{2}|^2$)이고 결과가 1일 확률이 50%이다. q0의 중첩 상태와 q1의 $|0\rangle$ 상태를 CNOT 게이트로 묶으면 얽힘 상태가 되는데 다음과 같다.

$$\frac{1}{\sqrt{2}}\,|00\rangle + \frac{1}{\sqrt{2}}\,|11\rangle \tag{4}$$

여기서 이 Bell 상태를 측정하면 00과 11이 나올 확률은 각각 50%이다. 따라서 01과 10이 나올 확률은 각각 0%이다. 측정된 값은 고전 레지스터(Classical Register)인 c에 저장되어 출력된다.

우리는 상태 $|\psi\rangle$가 다음과 같이 표현될 때 단순곱 상태(product state)에 있다고 말한다.

$$|\psi\rangle = |\psi_1\rangle\,|\psi_2\rangle \tag{5}$$

여기서 $|\psi_1\rangle$과 $|\psi_2\rangle$는 임의의 상태이다. 얽힘 상태란 이렇게 단순곱 상태가 아닌 상태를 말한다. 예로 얽힘 상태 중의 하나인 Bell 상태는 다음과 같다.

$$\begin{aligned} &\frac{1}{\sqrt{2}}\,|00\rangle + \frac{1}{\sqrt{2}}\,|11\rangle\,, \quad \frac{1}{\sqrt{2}}\,|00\rangle - \frac{1}{\sqrt{2}}\,|11\rangle \\ &\frac{1}{\sqrt{2}}\,|01\rangle + \frac{1}{\sqrt{2}}\,|10\rangle\,, \quad \frac{1}{\sqrt{2}}\,|01\rangle - \frac{1}{\sqrt{2}}\,|10\rangle \end{aligned} \tag{6}$$

얽힘 상태를 만드는 CNOT 게이트는 제어 큐비트(control qubit)인 q0가 0일 때 대상 큐비트(target qubit)인 q1은

그대로 둔다. q0가 1일 때 q1은 반대로 뒤집는다. 따라서 CNOT 게이트는 다음과 같이 작용한다.

$$|00\rangle \to |00\rangle\,; \quad |01\rangle \to |01\rangle\,; \quad |10\rangle \to |11\rangle\,; \quad |11\rangle \to |10\rangle\,. \tag{7}$$

정리하면 계의 상태가 처음에는 $|00\rangle$이었다가 Hadamard 게이트를 적용하면 상태는 $(|00\rangle + |10\rangle)/\sqrt{2}$으로 된다. q0의 상태를 측정하여 0 또는 1일 때 얻는 확률은 모두 50%이다. 이후 CNOT 게이트를 적용하면 $(|00\rangle + |11\rangle)/\sqrt{2}$의 최종 상태가 되는데 q0가 0일 때 q1을 측정하면 우리는 결과 값으로 0을 100%로 얻게 되고 마찬가지로 q0가 1일 때 q1이 100%로 1이 된다. 즉, 0과 0 그리고 1과 1이 서로 얽혀 있는 것이다. 마치 두 개의 동전을 동시에 던졌을 때 한쪽이 앞면(0)이 나오면 다른 쪽도 앞면(0)이 나오고 한 쪽이 뒷면(1)이 나오면 다른 쪽도 뒷면(1)이 나오는 상황이다. 두 동전이 서로 얽혀 있는 것이다. Figure 1은 그와 같은 얽힘 상태를 양자 회로를 통해 구현하는 것으로 독자도 IBM Quantum Experience를 이용하여 이와 같은 양자 얽힘 상태를 직접 구현할 수 있다.

본 논문에서는 $(|00\rangle + |11\rangle)/\sqrt{2}$의 Bell 상태를 축약된 네 가지 경우에 따라 구현하는 쉬운 방법을 소개하여 양자 계산을 시작하려는 학생들에게 도움을 주려고 한다. 유튜브(YouTube) [15]에서 qiskit tutorial을 검색하여 Abraham Asfaw의 Qiskit Foundations – Coding with Qiskit Season 1의 Episode 9개 동영상들을 시청하면 많은 도움이 될 것이다.

II. 본 론

우선 PC에 파이썬 언어를 설치하고 Jupyter 노트북 [16] 환경을 만들어야 한다. 이를 위해서 Anaconda [17]를 다운로드 받아 PC에 설치한다. Anaconda는 파이썬 언어를 기반으로 Jupyter 노트북을 제공하며 파이썬의 유용한 라이브러리인 numpy, scipy, matplotlib들을 함께 설치한다. 설치가 완료되면 Fig. 2와 같은 메뉴가 PC에 만들어진다. 여기서는 Windows 운영 체제에 대한 것만 설명하도록 한다. 파이썬 언어는 버전 3으로 설치가 된다.

다음으로 qiskit 패키지를 설치해야 한다. Qiskit을 설치하는 가장 좋은 방법은 파이썬 패키지 관리자(pip)를 사용하는 것이다. Figure 2의 3번째 줄에 있는 Anaconda Prompt (anaconda3)을 선택하여 실행한다. Figure 3과 같은 윈도우 창이 열린다. 커서가 있는 곳에서 다음과 같은 명령을 키보드로 입력하여 [enter] 키를 누른다.

pip install qiskit

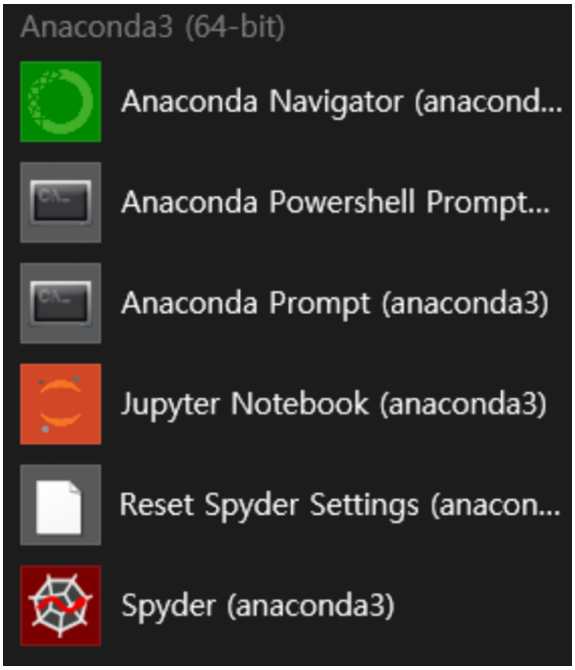


Fig. 2. (Color online) Menu of Anaconda on PC. The icon of Anaconda Prompt (anaconda3) is used to install qiskit and pylatexenc. The icon of Jupyter Notebook (anaconda3) is used to open the Jupyter notebook.

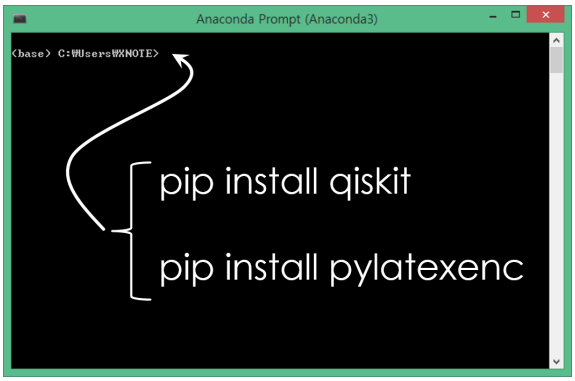


Fig. 3. (Color online) Window of Anaconda Prompt. Enter "pip install qiskit" and "pip install pylatexenc" at the prompt.

Qiskit을 설치하는 동안 얼마의 시간이 소요된다. 설치가 완료되면 같은 윈도우에서 다음과 같이 pylatexenc 도 함께 설치하는 것이 좋다. 이는 Jupyter 노트북에서 양자 회로 다이어그램을 보기 좋게 그려내는 역할을 한다.

pip install pylatexenc

Qiskit을 프로그래밍 하는데 있어서 선호하는 방법은 Jupyter 노트북을 사용해 파이썬 환경 내에서 코딩하는 것이다. Figure 2의 4번째 줄에 있는 Jupyter Notebook (anaconda3)을 선택하여 실행한다. Figure 4와 같은 윈도우 창이 열린다. Jupyter 노트북은 파이썬 입력 코드와 실행 결

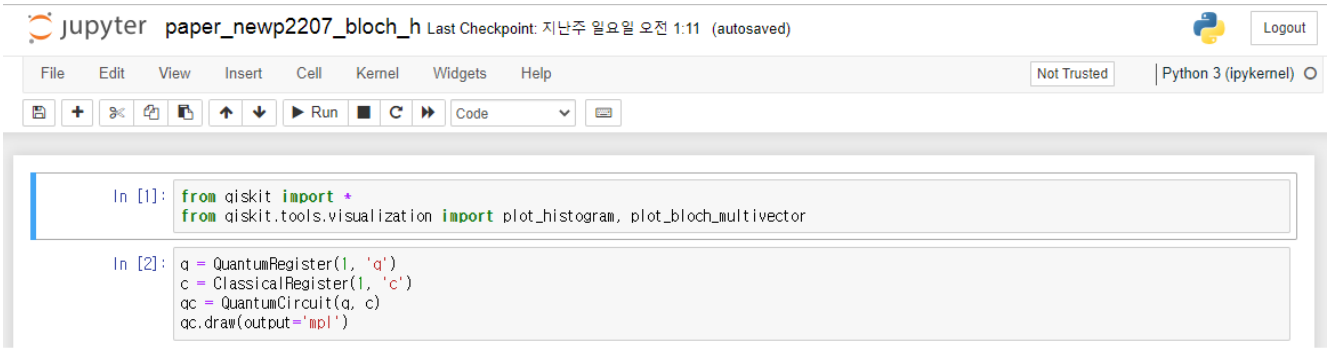


Fig. 4. (Color online) The layout of the Jupyter notebook. After the input of the first cell (In [1]) press [shift]+[enter] of the keyboard to execute the cell. It is the same case as that of the notebook of the Mathematica software [18].

과를 모두 한 곳에서 볼 수 있는 편리한 기능을 가지고 있다. Mathematica 소프트웨어에서 쓰는 노트북과 같은 개념이라고 생각하면 된다 [18]. 여기서도 Jupyter 노트북의 셀에 명령을 입력한 후 실행을 위해서 키보드의 [shift]+[enter] 키를 동시에 누른다. 이제 qiskit을 사용할 준비가 완료되었다.

다음으로는 네 가지 경우에 따른 축약된 버전들을 소개한다.

1. 버전 1: Bloch 구의 계산

Figure 5에는 첫 번째 버전의 파이썬 프로그램과 그 결과들이 나타나 있다. 프로그램 순서는 네 가지 경우 즉, 버전 1에서 4까지 공히 다음과 같이 다섯 부분으로 나누어진다.

(1) In [1]: 패키지를 불러 옴.
(2) In [2]: 양자 회로를 구성함.
(3) Out [2]: 양자 회로 다이어그램을 출력함.
(4) In [3]: Backend를 결정하고 Job을 보내어 계산함.
(5) Out [3]: 계산 결과를 출력함.

버전 1에서는 qiskit을 사용해 큐비트의 Bloch 구를 가시화하는 방법에 대해서 알아본다. Bloch 구는 크기가 1인 반경을 갖는 3차원 구체이며 구체 위의 임의의 양자 상태는 다음과 같이 표현된다.

$$|\varphi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\theta}\sin\left(\frac{\theta}{2}\right)|1\rangle \qquad (8)$$

구의 중심에서 화살표로 가장자리를 가리키는 벡터로 큐비트를 표현할 수 있다. 큐비트가 중첩일 경우에는 즉, Eq. (3)의 $(|0\rangle + |1\rangle)/\sqrt{2}$ 상태일 경우, 화살표는 정확히 x 방향을

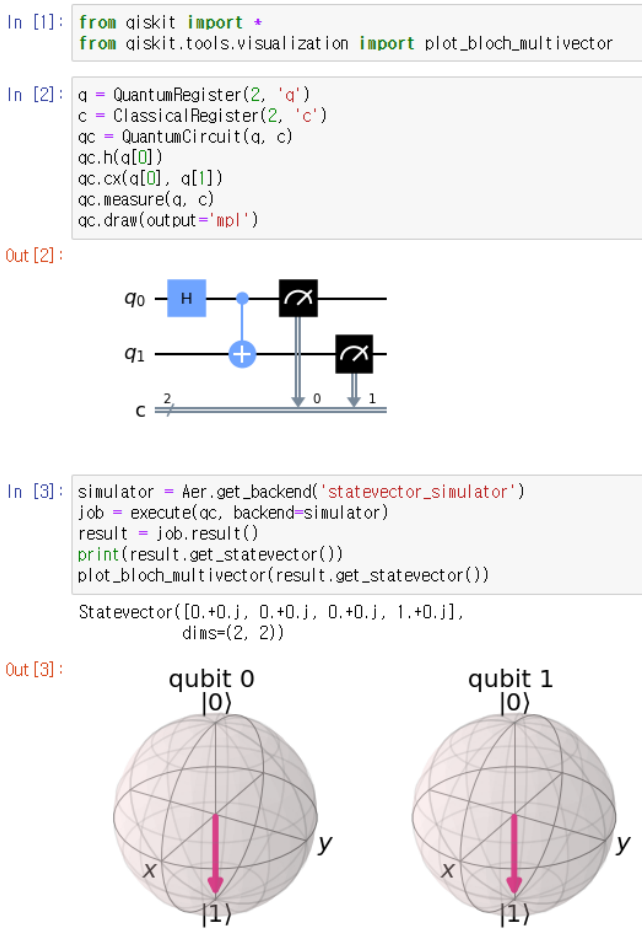


Fig. 5. (Color online) Python program for executing quantum instructions on PC simulator to plot the Bloch sphere of the Bell state.

가리킨다. 이와 같이 Bloch 구에 큐비트를 가시적으로 나타내는 것은 큐비트를 이해하는데 많은 도움이 된다. 특히 큐비트를 양자 게이트로 조작하고 그 결과를 보는 과정에서 매우 유용하다. Figure 5의 Out [3]:에는 Bloch 구가 2개 나타나 있다. 둘 다 $|1\rangle$의 상태를 가리키고 있다. 즉, $|11\rangle$인

상태를 보여주고 있는 것이다. 계산을 돌리다 보면 둘 다 $|0\rangle$의 상태를 가리키는 $|00\rangle$인 상태도 볼 수 있다. 앞에서 보았듯이 $|01\rangle$과 $|10\rangle$인 상태는 나타나지 않는다.

Figure 5의 In [1]:에 쓰여 있는

from qiskit import *

는 qiskit 모듈에서 모든 클래스와 메소드를 가지고 온다는 의미이다. 그리고

from qiskit.tools.visualization import plot_bloch_multvector

는 qiskit.tools.visualization 모듈에서 Bloch 구를 출력하는 함수를 불러 온다.

Figure 5의 In [2]:에서 우선 양자 레지스터 (QuantumRegister), 고전 레지스터(ClassicalRegister), 양자 회로(QuantumCircuit)를 설정한다. 양자 레지스터의 입력 변수는 2와 'q'이다. 이는 양자 레지스터가 두 개의 큐비트를 포함하고 변수가 q임을 나타낸다. 고전 레지스터의 입력 변수는 2와 'c'이다. 이는 고전 레지스터가 두 개의 비트를 포함하고 변수가 c임을 나타낸다. 양자 회로 클래스는 두 개의 큐비트를 포함한 양자 레지스터와 두 개의 비트를 포함한 고전 레지스터를 가진 회로를 만든다. 이제 양자 게이트를 각각의 큐비트에 작용할 준비가 되었다. 양자 게이트는 양자 상태를 변경하기 위하여 양자 레지스터에 작용한다. 양자 회로의 메소드로 h(), cx(), measure(), draw()가 쓰이고 있다. Bell 상태를 만드는 양자 회로를 구성한다. 여기서 h()는 Hadamard 게이트로 첫 번째 큐비트인 q[0]에 작용한다. H 게이트로 알려진 Hadamard 게이트는 앞에서 보았듯이 Eq. (3)의 중첩 상태를 만든다. cx()는 CNOT 게이트로 얽힘 상태를 만들기 때문에 양자 계산에서 매우 중요한 역할을 한다. 얽힘 상태는 각각의 큐비트가 개별적으로는 의미를 가질 수 없고 그룹으로만 의미를 갖는다. 두 번째 큐비트는 대상 큐비트로 첫 번째 큐비트인 제어 큐비트에 따라 동작한다. 첫 번째 큐비트 q[0]는 제어 큐비트로 동작하고 변하지 않는다. 첫 번째 큐비트가 $|1\rangle$인 경우, NOT 동작은 두 번째 큐비트 q[1]에 적용된다. 첫 번째 큐비트가 $|0\rangle$라면, 두 번째 큐비트에는 어떠한 변화도 없다. 이제 결과를 측정한다. measure()가 이를 수행한다. 이 메소드는 측정을 위해 양자 레지스터 q와 결과를 담을 고전 레지스터 c를 입력으로 취한다. 다음은 양자 회로를 출력하는 draw() 메소드이다. 양자 회로 다이어그램은 양자 회로를 가시화하는 방법이다.

Figure 5의 Out [2]:는 출력의 결과물이며 Bell 상태를 구현하는 회로로 Fig. 1과 같다. 일반적으로 양자 회로 다이어그램의 맨 왼쪽에는 q0와 q1과 같이 여러 개의 $|0\rangle$ 큐비트로

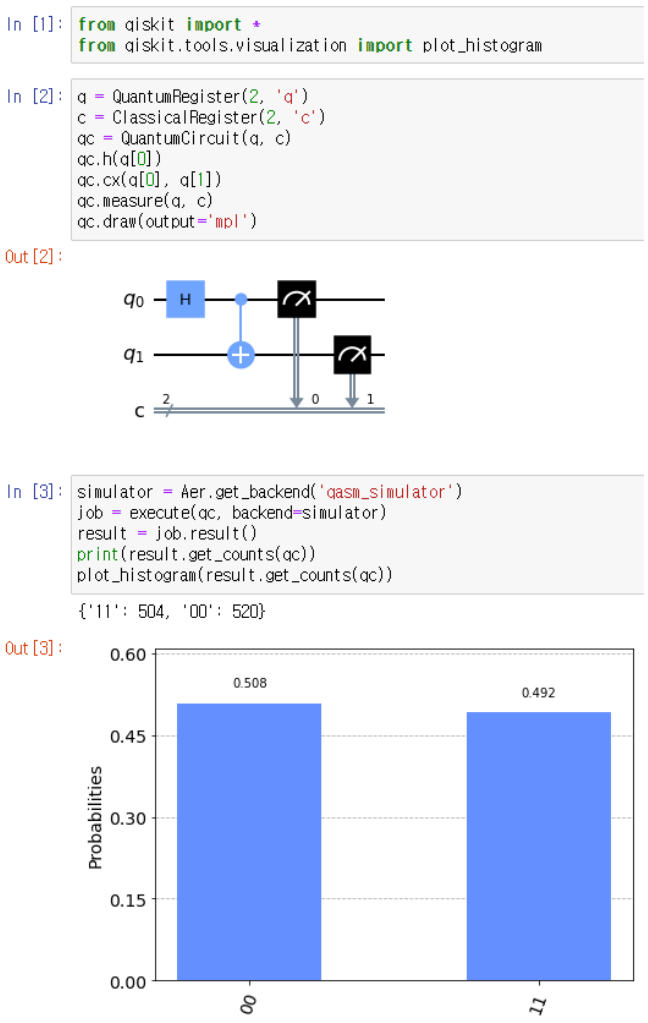


Fig. 6. (Color online) Python program for executing quantum instructions on PC simulator to plot the histogram of the Bell state.

초기화 된 양자 레지스터가 오며 이를 시작으로 전선처럼 보이는 선들이 연결된 모습을 보여준다. 아래쪽에는 c 즉, 고전 레지스터 두 개가 있다. Hadamard 게이트는 H가 쓰여진 상자로 표시되어 있다. CNOT와 같은 2개의 큐비트 게이트는 제어 큐비트와 대상 큐비트가 전선으로 연결되어 있다. 제어 큐비트는 작은 파란색 원으로 표시되어 있고 대상 큐비트는 크고 십자가가 표시된 원형으로 되어 있다. 다음으로는 두 개의 큐비트에 걸쳐 있는 측정 게이트이다. 이의 결과는 모두 고전 레지스터인 c로 보내어진다.

Figure 5의 In [3]:의 첫 번째 나오는 명령은 backend를 설정한다. 여기서는 Bloch 구를 계산하는 'statevector_simulator'가 쓰여졌다. 그 다음은 이렇게 정한 backend를 가지고 양자 회로 qc를 계산한다. 즉, backend에서 실행할 job을 보내고 그 결과 값 result.get_statevector()를 받는다.

Figure 5의 Out [3]:에는 버전 1의 결과물인 Bloch 구가 출력되어 있다.

2. 버전 2: qasm_simulator를 이용한 histogram의 계산

여기서는 앞의 버전 1의 경우와 모든 것이 같으나 backend로 qasm_simulator를 쓰고 그 결과 값으로 histogram을 출력하는 것이 다르다. Figure 6의 In [1]:의 두 번째 줄에 있는

from qiskit.tools.visualization import plot_histogram

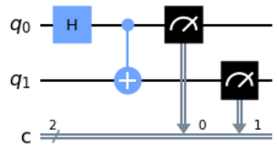
이 나와 있는데 qiskit.tools.visualization 모듈에서 histogram을 출력하는 함수를 불러 온다. Figure 6의 In [3]:에 있는 backend로 qasm_simulator를 설정했는데 이는 실제 하드웨어 양자 컴퓨터가 아닌 PC에서 돌아가는 시뮬레이터이다. 클라우드에 연결하지 않고도 작동한다. 결과값으로 result.get_counts(qc)를 가져오는데 여기서는 기본으로 1024 shots를 계산하였다. 00이 520번 그리고 11이 504번 나왔다. 이를 확률로 계산하면 histogram에서 볼 수 있듯이 각각 50.8%와 49.2%가 된다. 이 상황을 다시 동전 던지기에 비유하자면, 동전이 땅에 떨어져서 우리가 그 결과 값으로 앞면(0) 또는 뒷면(1)을 보는 행위를 측정이라고 한다. 동전 던지기를 1번 수행하여 측정하면 이를 1 shot이라고 하고 따라서 여기에서는 1024번 동전 던지기를 수행하여 통계를 내서 이를 histogram으로 나타낸 것이다. 즉, 결과 값이 00이 520번 그리고 11이 504번 측정된 것이다. 이 값들을 1024로 나누어 100을 곱하면 확률이 나온다. shots는 결국 통계를 내기 위한 실험의 반복 횟수라고 볼 수 있다.

Backend로 버전 1에서는 statevector_simulator를 사용하고 버전 2에서는 qasm_simulator를 사용하였는데 그 차이는 다음과 같다. Qiskit의 Aer backend에는 네 가지 시뮬레이터가 있는데, Qasm simulator, Statevector simulator, Unitary simulator, Pulse simulator 가 그것이다. statevector_simulator는 이름이 말해 주듯이 1 shot 이후에 회로의 최종 state vector를 제공한다. 따라서 회로의 최종 Bloch 구를 출력할 수 있다. 반면에 qasm_simulator는 여러 번의 shots (기본 1024 shots)를 실행하여 측정된 상태의 횟수를 제공한다. 따라서 회로의 최종 확률이 기록되어 있는 histogram을 출력할 수 있다.

```
In [1]: from qiskit import *
        from qiskit.tools.visualization import plot_histogram
        from qiskit.tools.monitor import job_monitor

In [2]: q = QuantumRegister(2, 'q')
        c = ClassicalRegister(2, 'c')
        qc = QuantumCircuit(q, c)
        qc.h(q[0])
        qc.cx(q[0], q[1])
        qc.measure(q, c)
        qc.draw(output='mpl')
```

Out [2]:




```
In [3]: IBMQ.load_account()
        provider = IBMQ.get_provider('ibm-q')
        qcomp = provider.get_backend('ibmq_manila')
        job = execute(qc, backend=qcomp)
        job_monitor(job)

        Job Status: job has successfully run

In [4]: result=job.result()
        print(result.get_counts(qc))
        plot_histogram(result.get_counts(qc))

        {'00': 1913, '01': 98, '10': 107, '11': 1882}
```

Out [4]:

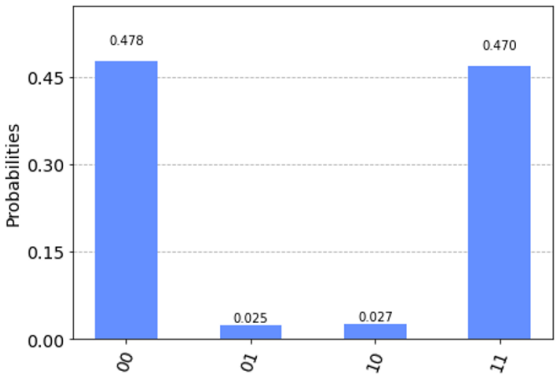


Fig. 7. (Color online) Python program for executing quantum instructions on the real IBM quantum computer at the system "ibmq_manila" to plot the histogram of the Bell state.

3. 버전 3: 실제 하드웨어에서의 계산

실제 하드웨어가 있는 IBM Quantum Experience에 접속하기 위해서는 홈페이지에서 자신의 계정(Account)을 만들고 거기서 API token을 생성하여 가져와야 한다. 이 token을 Jupyter 노트북에 입력하고, 나중을 위해서 다른 곳에 기록해 두었다가 필요할 때 복사해서 사용한다. Jupyter 노트북에서는 다음 명령과 같이 실행한다.

IBMQ.save_account("INSERT_YOUR_API_TOKEN_HERE")

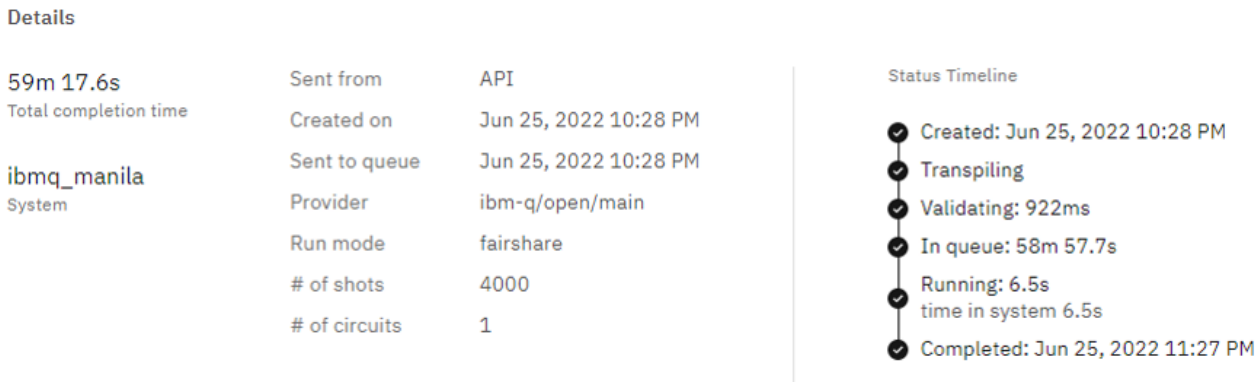


Fig. 8. (Color online) Details of the job sent to the IBM system "ibmq_manila" of the Python program in Fig. 7.

코드에 API token을 복사하여 붙여 넣는다. 한 번만 실행하면 된다.

Figure 7의 In [1]:의 세 번째 줄에 나와 있는 것은 qiskit.tools.monitor 모듈에서 job_monitor 함수를 불러오는 명령이다. Figure 7의 In [3]:의 첫 번째 줄은 save_account() 했던 API 토큰을 load_account() 해서 불러오는 것이다. provider 는 'ibm_q'이고 backend 는 'ibmq_manila'이다. 실제 양자 컴퓨터 하드웨어인 'ibmq_manila'에 job을 보내고 계산 상황을 모니터링하기 위해 job_monitor(job)을 실행한다. 처음에는 계산 순서가 몇 번째 queue에 있는지 보여주다가 최종 Job Status로 job has successfully run 이 나오면 계산이 성공적으로 완료되었다는 것을 알려준다. 다음과 같은 순서로 출력된다.

> Job Status: job is being validated
> Job Status: job is queued (. . .)
> Job Status: job has successfully run

Figure 8은 IBM Quantum Experience의 내 계정에 나와 있는 job에 대한 상세한 내용을 나타낸 것이다. Job에 소요된 총 시간은 59분 17.6 초이고 System은 ibmq_manila 이다. Queue에서 기다리는데 58분 57.7초가 소요되었으며 실제 계산에는 6.5초밖에 걸리지 않았다. 총 4000 shots 를 계산하였다. 2022년 6월 25일 현재 IBM Quantum Experience에서 운영하는 System 상황은 Table 1에 나와 있다. 총 22개의 양자 컴퓨터가 가동 중에 있다. 이외에 ibmq_qasm_simulator가 있는데 이는 원격으로 Emulator에 접속하는 것이다.

Figure 7의 In [4]:는 버전 2와 같다. 다만 결과에서 $|01\rangle$과 $|10\rangle$에서 각각 2.5%와 2.7% 나오는 것은 실제 양자 컴퓨터이기 때문에 noise에 의한 것으로 추측할 수 있다.

Table 1. Available IBM quantum computers of 22 systems with their qubits at the stand of June 25, 2022.

System	Qubits	System	Qubits
ibm_washington	127	ibmq_guadalupe	16
ibmq_brooklyn	65	ibm_perth	7
ibm_ithaca	65	ibm_lagos	7
ibmq_kolkata	27	ibm_nairobi	7
ibmq_montreal	27	ibm_oslo	7
ibmq_mumbai	27	ibmq_jakarta	7
ibm_cairo	27	ibmq_manila	5
ibm_auckland	27	ibmq_quito	5
ibm_hanoi	27	ibmq_belem	5
ibmq_toronto	27	ibmq_lima	5
ibm_peekskill	27	ibmq_armonk	1

4. 버전 4: OpenQASM을 이용한 양자 회로의 작성

지금까지 qiskit을 이용하여 양자 회로를 구성하였다. 그런데 종종 OpenQASM으로 양자 회로를 작성하는 경우가 있다. 여기서는 OpenQASM 프로그램을 qiskit 코드로 통합하는 경우를 살펴본다. Figure 9의 In [2]:에서는 문자열로부터 OpenQASM 코드를 불러와 양자 회로를 구성하는 예가 나와 있다. 세 개의 인용부호(" " ")로 둘러싸인 qasm_string 변수에 코드를 입력하면 된다. 모든 프로그램은 OPENQASM 2.0;으로 시작해야 한다. IBM Quantum Experience에서 연산을 할 때 include "qelib1.inc"; 헤더가 추가된다. 다음 줄은 크기가 2이고 q로 이름 지은 양자 레지스터를 선언한다. 다음으로 2 개의 고전 레지스터를 c로 선언한다. 이후는 Hadamard 게이트와 CNOT 게이트를 차례로 적용한다. 마지막으로 측정을 하면 양자 회로의 구성은 끝난다. 이렇게 완성한 OpenQASM 코드 문자열을

qc = QuantumCircuit.from_qasm_str(qasm_string)

으로 넘기면 된다.

Qiskit이 파이썬을 기반으로 한 고급 언어에 속한다면 OpenQASM은 고전적인 어셈블리 언어와 비슷하다. OpenQASM은 고급 언어인 파이썬 코드를 저수준으로 변환해서 표현한 것으로 IBM Quantum Experience와 오픈소스 커뮤니티 간 협업의 결과물이다. IBM Quantum Composer에서 Graphics User Interface로 작성된 양자 회로는 OpenQASM으로 실시간 출력됨으로 이를 qiskit에 가져와 버전 4처럼 실행할 수 있다.

III. 결 론

양자 계산의 일 예로 Bell 상태를 계산하였다. IBM Quantum Experience에서 양자 계산을 실행하기 위해서 PC에 anaconda와 qiskit을 설치하였다. 파이썬 기반으로 Jupyter 노트북을 사용하여 모든 양자 프로그램을 실행하였다. 학생들에게 유익한 네 가지 버전의 qiskit 프로그램을 선보였다.

버전 1: Bloch 구의 계산
버전 2: qasm_simulator를 이용한 histogram의 계산
버전 3: 실제 하드웨어에서의 계산
버전 4: OpenQASM을 이용한 양자 회로의 작성

양자 계산을 시작하는 학생들에게 많은 도움이 되기를 바란다.

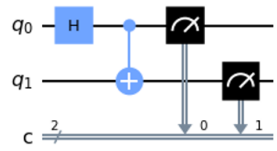
REFERENCES

[1] R. P. Feynman, Int. J. Theor. Phys. **21**, 467 (1982).
[2] https://webzine.kps.or.kr/?p=4&idx=25
[3] F. Arute *et al.*, Nature **574**, 505 (2019).
[4] IBM Quantum Experience homepage, https://quantum-computing.ibm.com/
[5] https://aws.amazon.com/ko/braket/
[6] https://azure.microsoft.com/ko-kr/
[7] https://qcenter.kr
[8] Qiskit homepage, https://qiskit.org
[9] Qiskit textbook homepage, https://qiskit.org/learn/
[10] Python homepage, https://www.python.org
[11] G. F. de Jesus *et al.*, arXiv:2101.11388 (2021).
[12] C. C. Moran, *Quantum Computing with IBM QX* in Korean (Acorn, Seoul, 2020).


```
In [1]: from qiskit import *
        from qiskit.tools.visualization import plot_histogram

In [2]: qasm_string = """
        OPENQASM 2.0;
        include "qelib1.inc";
        qreg q[2];
        creg c[2];
        h q[0];
        cx q[0], q[1];
        measure q[0] -> c[0];
        measure q[1] -> c[1];
        """
        qc = QuantumCircuit.from_qasm_str(qasm_string)
        qc.draw(output="mpl")

Out[2]:
```




```
In [3]: simulator = Aer.get_backend('qasm_simulator')
        job = execute(qc, backend=simulator)
        result = job.result()
        print(result.get_counts(qc))
        plot_histogram(result.get_counts(qc))

        {'00': 518, '11': 506}

Out[3]:
```

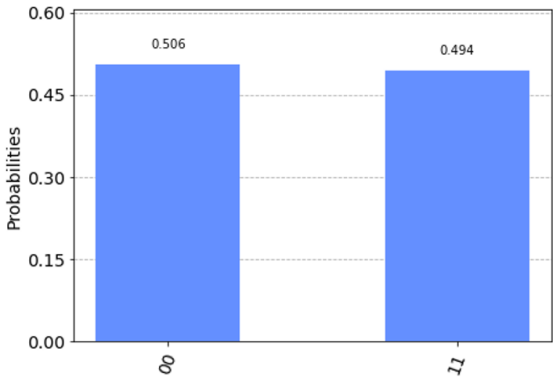


Fig. 9. (Color online) Python program for executing quantum instructions written by OpenQASM on PC simulator to plot the histogram of the Bell state.

[13] V. Silva, *Practical Quantum Computing for Developers* in Korean (Acorn, Seoul, 2019).
[14] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, 2016).
[15] YouTube homepage, https://www.youtube.com
[16] Jupyter homepage, https://jupyter.org
[17] Anaconda homepage, https://www.anaconda.com
[18] Mathematica homepage, https://www.wolfram.com/mathematica/