量子コンピューターを使ってみよう
—QISKitを用いた量子プログラミングの紹介—

今道　貴司，井床　利生，ルディー・レイモンド

中小規模の量子コンピューターの研究開発が近年活発に行われており，IT 各社や研究機関は従来のコンピューターの限界を超える量子コンピューターの早期実現を目指している．ノイズがあり，量子ビットの数や量子ビット間の接続関係も限定的なこの量子コンピューターを利用するためのソフトウェア開発キット (SDK) の研究開発も同様に急ピッチで行われている．本記事では代表的な量子コンピューター用の SDK を紹介するとともに，IBM がオープンソースとして一般公開した SDK の QISKit を使ってどのように量子コンピューター IBM Q Systems 向けのプログラミングを行うかを述べる．

キーワード：量子コンピューター，プログラミング，QISKit

1. はじめに

量子アルゴリズムの研究は長らく理論的なものだったが，量子コンピューターの実機の研究が進んだことによって，ノイズ付きで量子ビット数が中小規模の量子コンピューター上で量子アルゴリズムを試すことができるようになってきた．また，その応用も量子化学や量子シミュレーションなどに変革をもたらす可能性が高いと言われており，量子コンピューターの開発競争がハードウェアとソフトウェアの両面で活発に行われている．

量子コンピューターと一口に言ってもいくつかの種類があるが，本稿では万能量子コンピューターの候補である量子ゲート型に焦点を当てる．万能量子コンピューターが実現されれば，Shor の素因数分解アルゴリズムや Grover の探索アルゴリズムなどの代表的な量子アルゴリズムの実行が可能になりインパクトが大きい．しかし，現存するゲート型量子コンピューターは計算中に生じるノイズに対処するための量子誤り耐性の実装が限定的で，上述したアルゴリズムの実行は不可能である．このような限定的なゲート型量子コンピューターは近似量子コンピューターと呼ばれる．説明の都合上，本記事では近似量子コンピューターも単に量子コンピューターと呼ぶ．

そのような制限があるにもかかわらず，50 量子ビットあたりの近似量子コンピューターが，従来のコンピューターでシミュレーションできるかできないかの境目だと言われている．そのため，現在その程度の中規模の量子コンピューターの実現に向けて IT 各社や研究機関がしのぎを削っている．IBM は 2017 年末に 50 量子ビットのプロセッサーのプロトタイプの完成を発表しているが，Google や Intel も 49 量子ビットのプロセッサーの開発を発表している．また，Rigetti などのベンチャー企業も独自技術で量子コンピューターのハードウェアとソフトウェアの研究開発およびビジネスの開拓を進めている．

近似量子コンピューターが実際の問題を解くのにどれぐらい役に立つかは未知数だが，量子化学などの量子コンピューターの量子力学的な性質を直接利用する分野が近い将来の応用として期待されている．詳細は [1] を参照してほしい．制限があるものの量子コンピューターがいち早く一般公開されることによって，量子ア

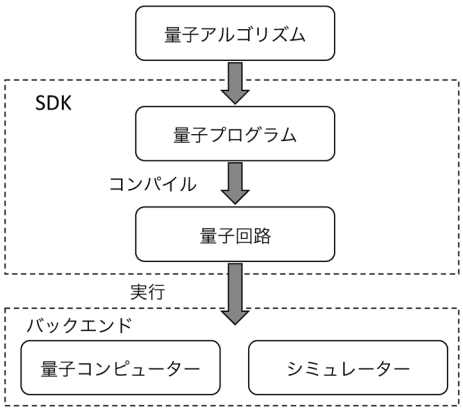

図 1　量子コンピュータを用いた計算の流れ

いまみち　たかし，いとこ　としなり，ルディー・レイモンド
IBM 東京基礎研究所
〒 103–8510　東京都中央区日本橋箱崎町 19–21
imamichi@jp.ibm.com
itoko@jp.ibm.com
rudyhar@jp.ibm.com

表 1　代表的な SDK の特徴（執筆時点）

SDK	Forest [2]	MS-QDK [3]	ProjectQ [4]	QISKit [5]
プログラミング言語	Python	C# / Python	Python	Python
対応バックエンド	Simulator, Rigetti Forest	Simulator	Simulator, IBM Q	Simulator, IBM Q
量子プログラミング言語	Quil	Q#	DSL	OpenQASM

ルゴリズムの研究や量子プログラムの開発が促進されて将来的に量子コンピューターの応用が急速に広がる可能性がある．深層学習が，理論の面から不明なところが多々あるものの，実装の面でその有用性が示され普及している状況とまさに同じようになる可能性を量子コンピューターは秘めている．

量子コンピューターを用いた計算の流れを図 1 に示す．量子アルゴリズムを実装するには，開発者は専用のソフトウェア開発キット (SDK) を使ってプログラムを書き，SDK を用いてそのプログラムを量子回路，または，量子コンピューターの基本命令のシーケンスにコンパイルする．そして，その量子回路を量子コンピューターやシミュレーターといったバックエンドで実行することで実行結果を得る．この SDK の開発競争も現在活発に行われている．

本記事では，量子コンピューター用の代表的な SDK を簡単に紹介するとともに，量子コンピューターとシミュレーターの両方に対応していて利用実績も高い IBM Quantum Experience と QISKit を取り上げて，近似量子コンピューターのプログラミングを紹介する．最後に，量子プログラミングの現状と課題を簡潔にまとめる．なお，本稿は 2018 年 2 月末の時点の情報をもとに記述している．

2.　量子コンピューター向け SDK

量子コンピューターの実機やシミュレーター技術の進展に伴って，それらの計算能力を最大限に引き出すためのソフトウェア技術や，量子プログラムの開発をサポートするためのソフトウェア技術，すなわちソフトウェアスタックにあたる SDK の重要性が増してきている．

本節では，まず表 1 に挙げる四つの代表的な SDK について簡単に紹介する．本稿では量子計算を行うプログラム（量子プログラム）を記述する専用言語のことを量子プログラミング言語と呼ぶ．

ProjectQ [4] は，チューリッヒ工科大学 (ETH) を起点に開発されている Python ベースのオープンソースソフトウェア (OSS) のプロジェクトである．四つの SDK の中では最も早く 2016 年末に公開された．量子プログラムは Python コードに埋め込む形で DSL (Domain Specific Language) を用いて記述する．ProjectQ の特徴としては，「高性能のシミュレーター／エミュレーターを持つこと」，「量子コンピューター実機に（クラウドサービスを通じて）アクセスできること」，「コンパイルや最適化を行うモジュールの拡張が容易なように設計されていること」の三点が挙げられる [6]．これらの特徴は，後述の OSS のツールキットである Forest や QISKit でも同様に重視されている．

Forest [2] は，ハードウェア開発も手がけるベンチャー企業 Rigetti が提供している開発環境である．実機へのアクセスはパートナーに限定されているが，シミュレーターへのアクセスは公開されている．独自言語 Quil を用いて量子プログラミングを行うための pyQuil というオープンソースの Python ライブラリも提供している．

Microsoft Quantum Development Kit [3]（以下，MS-QDK）は，Microsoft が提供する SDK である．独自の量子プログラミング言語である Q#で量子プログラムを記述し，それらを呼び出し外部と連携する部分は C#または Python で記述するというスタイルで開発する．統合開発環境である Visual Studio に統合されていてリッチなデバッグ環境も用意されている．

QISKit (Quantum Information Software Kit) [5] は，IBM が中心となって開発している OSS の SDK である．次節以降で詳しく紹介する．

上記 SDK 以外にもオランダに拠点を置く研究センター QuTech が開発している qc-toolkit など多数の研究グループが現在進行形で開発に取り組んでいる．

本稿では SDK に焦点を当てるが，たとえば，オープン量子系のダイナミクスをシミュレートするための OSS である QuTiP [7] や，量子化学計算用に開発された Python ライブラリの OpenFermion [8] など，特定のアプリケーションに特化したライブラリもある．

量子プログラミング言語は本稿の対象外とするが，魅力的な研究分野の一つである．たとえば Quipper [9] など関数型の量子プログラミング言語も提案されている．

以降では，上記 SDK の一つである QISKit，および，ウェブブラウザ上でプログラミングを行える IBM Quantum Experience に注目し，執筆時点で無償公開

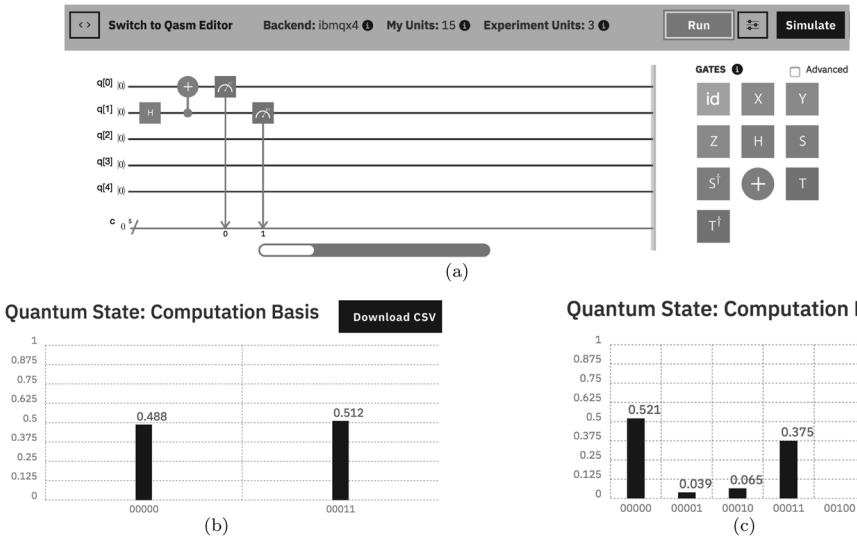

図 2　Composer のスクリーンショット．(a) ゲートの配置図，(b) シミュレーション結果，(c) ibmqx4 での実行結果

されている唯一の実機である IBM Q Systems 上で量子プログラムを実行する手順を紹介していく．

3. IBM Quantum Experience と QISKit の概要

IBM は IBM Q というイニシアチブで量子コンピューターの研究開発を行っており，その量子コンピューターのシステムは IBM Q Systems と呼ばれている．IBM Q Systems のプログラミング環境には，ウェブブラウザ上でプログラミングをする IBM Quantum Experience（以下 QX）と，Python 用の SDK の QISKit がある．

3.1 IBM Quantum Experience

QX [10] は，量子コンピューターや量子プログラミングの教材，プログラミング環境，ユーザーのコミュニティなどを備えたウェブサイトである．QX 上で，ユーザーは「Composer」と呼ばれるツールを使って視覚的に 1 量子ビットの回転ゲートや 2 量子ビットの Controlled-NOT (CNOT) ゲートを配置したり，「Qasm Editor」で OpenQASM [11] というアセンブリ言語を記述してプログラミングを行うことができる．執筆時点で，QX では ibmqx2 や ibmqx4 と呼ばれる 5 量子ビットの量子コンピューターの実機と，20 量子ビットまで対応したシミュレーターのいずれかでプログラムを実行することができる．理論的にはあらゆる量子アルゴリズムは上述の 1 量子ビットの回転ゲートと 2 量子ビットの CNOT ゲートで構成できるが，QX ではユーザーがそのような構成を自力で行わなければならない．なお，ユーザーはソフトウェアの開発に当たって，まず QX でアカウントを作成する必要がある．

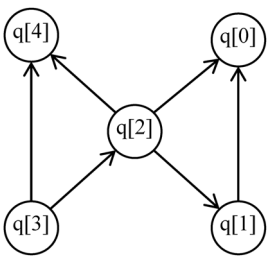

図 3　ibmqx4 の量子ビットの接続関係

図 2(a) はベル状態と呼ばれる $(|00\rangle + |11\rangle)/\sqrt{2}$ の状態を生成するプログラムを Composer 上で実装したものである．1 量子ビットゲートのアダマールゲート (H) を q[1] に置き，2 量子ビットの CNOT ゲートのコントロールビットを q[1] に，ターゲットビットを q[0] に置いて回路を構成している．なお，アダマールゲートは量子重ね合わせを，CNOT ゲートは量子もつれを実現する際に用いる代表的なゲートである．

2 量子ビットの CNOT ゲートは，実機上では必ずしも任意の量子ビット間に置くことができるわけではなく，実機固有の量子ビットの接続関係に従う．Composer でプログラムを書く際は，この量子ビット間の接続関係を満たすような場所にしか CNOT ゲートを配置できないようになっている．たとえば，ibmqx4 の量子ビットの接続関係は図 3 のようになっており，矢印 (q[1]→q[0]) の q[1] をコントロールビット，q[0] をターゲットビットとする CNOT ゲートは配置できる


```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[2];
creg c[2];
h q[1];
cx q[1], q[0];
measure q[0] -> c[0];
measure q[1] -> c[1];
```

図 4　図 2(a) の回路と等価な OpenQASM のプログラム

が，矢印が逆の q[0] をコントロールビット，q[1] をターゲットビットとする CNOT ゲートや，矢印のない q[0] をコントロールビット，q[4] をターゲットビットとする CNOT ゲートは Composer 上で配置できない．図 2(a) は ibmqx4 の接続関係を満たしている．

IBM Q Systems は近似量子コンピューターなので，各ゲートや読み出しなどでノイズが生じる．図 2(b) と (c) は，それぞれシミュレーターと ibmqx4 でプログラムを，1,024 回実行した結果の確率の棒グラフである．ビットパターンの下二桁が操作した 2 量子ビット（q[0] と q[1]）に対応している．シミュレーターの結果では “00000” と “00011” のみがほぼ同じ確率で観測されているが，ibmqx4 の結果では “00000” が観測される確率は “00011” の確率よりも若干高めである．また，“00001” と “00010” がわずかながら観測されている．これらが実機のノイズの影響で，特に “00000” が高めの確率で観測されることは量子ビットがエネルギーレベルの最も低いゼロ状態に収縮する傾向に起因する．このようなノイズの影響などにより，IBM Q Systems では実行できる量子回路の深さ（ステップ数）に制限がある．

Composer 上で QasmEditor に切り替えることで OpenQASM 形式でのプログラミングも可能である．図 4 は図 2(a) の回路を OpenQASM 形式にしたものである．

3.2　QISKit

QISKit [5] は Python 用の SDK で，Python の機能を利用することで QX よりも複雑なプログラミングが可能である．たとえば，任意の量子状態を生成する回路を自動化する関数など，量子アルゴリズムを本格的に実装する場合に必要なライブラリが備わっている．また 16 量子ビットの実機の ibmqx5 や，32 量子ビットまで対応したシミュレーターの ibmqx_hpc_qasm_simulator へのアクセスも可能である．さらに，QISKit 自体が local_qasm_simulator（Python 製）と local_qiskit_simulator（C++ 製）の

```
from qiskit import QuantumProgram
import Qconfig
qp = QuantumProgram()
qp.set_api(Qconfig.APItoken,
           Qconfig.config['url'])
qr = qp.create_quantum_register('qr',2)
cr = qp.create_classical_register('cr',2)
qc = qp.create_circuit('Bell',[qr],[cr])
qc.h(qr[0])
qc.cx(qr[0], qr[1])
qc.measure(qr[0], cr[0])
qc.measure(qr[1], cr[1])
result = qp.execute('Bell', shots=1000,
            backend='ibmqx4', timeout=3600)
print(result.get_counts('Bell'))
```

図 5　QISKit を使ったベル状態を生成するプログラム例

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[2];
creg cr[2];
u2(0,3.14159265358979) q[1];
cx q[1],q[0];
measure q[0] -> cr[1];
measure q[1] -> cr[0];
```

図 6　QISKit が図 5 を ibmqx4 用に変換した結果の OpenQASM のプログラム

二種類を含んでいて，手元の PC でこれらのシミュレーターを実行することができる．

QISKit のユーザーは QX のアカウント作成後に，QISKit のインストールと設定が必要である．QISKit は Python 3.5 以上に対応しており，最も簡単なインストール方法は以下のとおりである．

```
$ pip install qiskit
```

ユーザーが QISKit を利用する際は QX のアカウントのページからトークンを Qconfig.py というファイルの指定の場所にコピーする必要がある[1]．なお，本記事中のスクリプトは QISKit のバージョン 0.4.9 を用いて実装した．

QISKit では内蔵のコンパイラーに各実機で実行可能な量子回路への変換を指定できるため，ユーザーは量子ビット間の接続関係を意識せずにプログラムを書くことができる．図 5 は QISKit で書いたベル状態を生成するプログラムの例[2] で，ibmqx4 向けに変換された OpenQASM の結果が図 6 である．元のプログラム

[1] インストールの詳細は QISKit のドキュメンテーションを参照してほしい．QIKit, install and setup. https://www.qiskit.org/documentation/install.html

では第 0 量子ビット (qr[0]) と第 1 量子ビット (qr[1]) がそれぞれコントロールとターゲットとして指定されているが，コンパイル後の図 6 では，ibmqx4 の量子ビットの接続関係を満たすために，コントロールとターゲットが入れ替えられていて，qr[0] が q[1] に，qr[1] が q[0] に対応するように変換されている．

実機の詳細な情報は，QISKit の API や QX のページ[3]から得られる．1 量子ビットゲートのエラー率，2 量子ビットゲートのエラー率，読み出しのエラー率，量子ビットの接続関係などのデータを見ることができる．

ちなみに，執筆時点で，IBM Q Systems の利用者は合計 7 万人以上，関連する研究論文も 60 本以上ある．なお，IBM は顧客向けに 20 量子ビット以上の量子コンピューターを含めた最新のシステムへのアクセス提供および研究開発のサポートを行っている．

4. QISKit を用いた量子プログラミングの実践例

QISKit には，Jupyter Notebook 形式で記述された多数のチュートリアルが付属している[4]．本節では，その中から 1 量子ビットを用いた Quantum Random Access Coding (Quantum RAC, QRAC) を紹介し，1 量子ビットの回転ゲートだけで量子重ね合わせを活用し，いかに量子的なアドバンテージが得られるか，最もシンプルな例を示す．

(m,n)-QRAC とは，m 古典ビットの情報を n 量子ビットに符号化する方法で，任意のビットを 1/2 より大きな確率で復号可能なものをいう．同様に，m 古典ビットを n 古典ビットに符号化する方法を，(m,n)-RAC という．興味深いことに，$(2,1)$-RAC は存在しないが，$(2,1)$-QRAC は存在する．また，$(3,1)$-QRAC は存在するが $(4,1)$-QRAC は存在しない [12]．一般に，$(2^{2n}-1,n)$-QRAC が存在することが知られている [13] が，$n \geq 2$ については，復元確率を最良にする QRAC を実際に構成する方法が知られていない組合せも多く，研究途上である．ちなみに，復号の成功確率が 1，つまり間違いを許さない場合は $n \geq m$ が必要で，量子ビットから得られる情報は古典ビットと同じであることが知られている．

[2] Windows 環境で本稿の例を実行できない場合，プログラム全体を「`if __name__=='__main__':`」の下に書く必要がある．

[3] IBM Q Experience devices, https://quantumexperience.ng.bluemix.net/qx/devices

[4] QISKit tutorials, https://github.com/QISKit/qiskit-tutorial

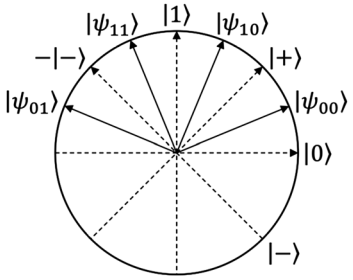

図 7　$(2,1)$-QRAC の $|0\rangle$-$|1\rangle$ 平面表現 [12]

以下では，まず $(2,1)$-QRAC の符号化および復号方法を天下り的に与え，次にそれを QISKit 上で実装し，シミュレーターおよび実機で実行する手順を示す．

まず，$(2,1)$-QRAC において，符号化は，2 ビットの情報 $00,01,10,11$ に対応する 1 量子ビットの量子状態（式 (1) の $|\psi_{00}\rangle, |\psi_{01}\rangle, |\psi_{10}\rangle, |\psi_{11}\rangle$）を作ることに相当する．

$$\begin{aligned} |\psi_{00}\rangle &= \cos(1\pi/8)\,|0\rangle + \sin(1\pi/8)\,|1\rangle\,, \\ |\psi_{10}\rangle &= \cos(3\pi/8)\,|0\rangle + \sin(3\pi/8)\,|1\rangle\,, \\ |\psi_{11}\rangle &= \cos(5\pi/8)\,|0\rangle + \sin(5\pi/8)\,|1\rangle\,, \\ |\psi_{01}\rangle &= \cos(7\pi/8)\,|0\rangle + \sin(7\pi/8)\,|1\rangle\,. \end{aligned} \tag{1}$$

また，復号化は，それら量子状態を巧妙に設計された基底で測定する（式 (2) の $E^i(i=1,2)$ がそれぞれ i ビット目の情報を取り出す基底）ことに相当する．

$$\begin{aligned} E^1 &= \{|0\rangle\langle 0|\,,\ |1\rangle\langle 1|\}, \\ E^2 &= \{|+\rangle\langle +|\,,\ |-\rangle\langle -|\}. \end{aligned} \tag{2}$$

ここで $|\pm\rangle = (|0\rangle \pm |1\rangle)/\sqrt{2}$ である．なお，ビット情報は左から順に数えるものとする．たとえば，10 の場合 1 ビット目を 1，2 ビット目を 0 とする．

これらは，Hayashi et al. [12] による図 7 によって，直感的に理解することができる．たとえば，情報 10 を符号化した $|\psi_{10}\rangle$ について，1 ビット目を E^1 で測定する場合，$|1\rangle$ と $|\psi_{10}\rangle$ のなす角が $\pi/8$ であることから，1 ビット目を 1 と正しく復元できる確率が $\cos^2(\pi/8) \approx 0.854$ である，といったことが見て取れる．

次に，$(2,1)$-QRAC を QISKit を用いて実装したプログラム例を図 8 に示す．符号化は，$|0\rangle$ に初期化された 1 量子ビット qr[0] に，パラメータを適当に調整した u3 ゲートを適用することで実現している．u3 ゲートは 1 量子ビットに対する任意のユニタリ演算を表現できる IBM Q Systems の基本ゲートの一つである [11]．ここでは，$|0\rangle$ に $\mathrm{u3}(\theta,0,0)$ を適用すると


```
from math import pi
from qiskit import QuantumProgram
import Qconfig
qp = QuantumProgram()
qp.set_api(Qconfig.APItoken,
           Qconfig.config['url'])
qr = qp.create_quantum_register("qr", 1)
cr = qp.create_classical_register("cr", 1)
# 回路生成
def gen_circuit(qp, pattern, position):
   qc = qp.create_circuit('QRAC',[qr],[cr])
   # 符号化
   coefs = {'00':1, '10':3, '11':5, '01':7}
   qc.u3(coefs[pattern]*pi/4, 0, 0, qr[0])
   qc.barrier()
   # 復号化
   if position == 'second':
       qc.h(qr[0])
   qc.measure(qr[0], cr[0])
   return qc
pattern = '01'  # 00 or 01 or 10 or 11
position = 'second'  # first or second
gen_circuit(qp, pattern, position)
# 実行
result = qp.execute('QRAC', shots=1000,
            backend='local_qasm_simulator')
print('Encode "{}" and measure the {} bit'.
       format(pattern, position))
print(result.get_counts('QRAC'))
```

図 8　(2, 1)-QRAC プログラム

$\cos(\theta/2)\,|0\rangle + \sin(\theta/2)\,|1\rangle$ が得られることを利用している．

また，QISKit では基底 E^1 での測定しか許されないため，基底 E^2 で測定を行いたい場合は，復号化部分の `if` 文にあるように，基底変換に相当するゲート（今回はアダマールゲート）を適用してから測定を行う．当プログラム例では，シミュレーター上で，2 ビット情報 01 を符号化し，二番目のビットを復号化するプログラムを 1,000 回実行して，結果を標準出力している．たとえば，以下のように '1' の観測数として $1000 \times 0.854 = 854$ に近い結果が得られる．

```
Encode "01" and measure the second bit
{'0': 137, '1': 863}
```

実機上で実行する場合は，`qp.execute()` の引数に，たとえば `backend='ibmqx4'` を設定する．すると，以下のように使用しなかった上位 4 量子ビットが 0 で埋まった結果が返ってくる．

```
Encode "01" and measure the second bit
{'00000': 178, '00001': 822}
```

QISKit チュートリアルには，本稿で取り上げた QRAC 以外にも，量子テレポーテーションや量子位相推定といった基本的な量子アルゴリズムから，古典コンピューターと量子コンピューターをハイブリッドで用いた量子化学計算や，乱数生成器として量子ビットを用いた海戦ゲームといった変わった応用まで，豊富な例題が集められているので，興味をもった方はぜひ調べてみてほしい．

5.　まとめと今後の展望

本稿では，量子コンピューターおよび量子プログラミングのための SDK の現状について整理し，QISKit を用いて実際に量子プログラミングを行う具体的な手順を紹介した．量子アルゴリズムを実装する場合のイメージが少しでも伝わっていれば幸いである．

量子コンピューターの研究開発は急速に進んでいるため，今後の動向については「全く予想がつかない」の一言に尽きるが，以下のようなことは言えるだろう．

量子コンピューターは古典コンピューターを完全に置き換える類の計算機ではなく，どちらかと言えば GPU アクセラレーターのように，特定の計算において古典コンピューターを上回る性能を発揮するタイプの計算機である．つまり，古典コンピューターと合わせて利用することでその性能を最大限に引き出せる．その意味で，古典と量子コンピューターをハイブリッドに利用するアルゴリズムへの注目度は高い．中でも，Variational Quantum Eigensolver (VQE) [14] のように近似量子コンピューターでも動作するハイブリッドアルゴリズムが最も応用に近いと目されている．本記事で紹介した SDK のほとんどに VQE とそれを利用するサンプルが付属しているのも偶然ではないだろう．

量子コンピューターは，今後とも長期的な研究を行ううえで非常に魅力的な分野であると言えよう．飛躍的な進歩につながるハードウェアはもちろんのこと，ソフトウェア，特に計算理論とプログラミング，についても研究課題は山積している．たとえば，量子コンピューターの計算能力と古典コンピューターの計算能力の本質的な違いがどこにあるのか，理論的な解明が急がれている．また，現在の量子プログラミング環境は，まるで古典コンピューターの黎明期にプログラムをアセンブラで書いていた頃のようである．高級言語に相

当するものをどう設計するか，優秀なコンパイラーおよび線形代数や探索などの基本的なライブラリをどのように開発するか，本稿で紹介したSDKはその解の探求におけるある種の試みであると言えよう．いずれ訪れるであろうハイプカーブの山も谷も乗り越えて，重ね合わせ状態にある量子コンピューターの未来が世の役に立つ側に収束する日が来ることを願ってやまない．

参考文献

[1] J. Preskill, “Quantum Computing in the NISQ era and beyond,” arXiv: 1801.00862, 2018.

[2] Rigetti, “Forest,” https://www.rigetti.com/forest（2018年3月26日閲覧）

[3] Microsoft, “Microsoft Quantum Development Kit,” https://www.microsoft.com/en-us/quantum/development-kit （2018年3月26日閲覧）

[4] ProjectQ, “ProjectQ,” https://projectq.ch/（2018年3月26日閲覧）

[5] QISKit, “Quantum Information Software Kit,” https://www.qiskit.org/（2018年3月26日閲覧）

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

[7] J. R. Johansson, P. D. Nation and F. Nori, “QuTiP: An open-source python framework for the dynamics of open quantum systems,” *Computer Physics Communications*, **183**, pp. 1760–1772, 2012.

[8] J. R. McClean, I. D. Kivlichan, D. S. Steiger, Y. Cao, E. S. Fried, C. Gidney, T. Häner, V. Havlíček, Z. Jiang, M. Neeley, T. O’Brien, I. Ozfidan, M. D. Radin, J. Romero, N. Rubin, N. P. D. Sawaya, K. Setia, S. Sim, M. Steudtner, W. Sun, F. Zhang and R. Babbush, “OpenFermion: The electronic structure package for quantum computers,” arXiv: 1710.07629, 2017.

[9] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger and B. Valiron, “Quipper: A scalable quantum programming language,” *ACM SIGPLAN Notices*, **48**, pp. 333–342, 2013.

[10] IBM, “IBM Quantum Experience,” https://quantumexperience.ng.bluemix.net/qx/experience（2018年3月26日閲覧）

[11] A. W. Cross, L. Bishop, J. Smolin and J. M. Gambetta, “Open quantum assembly language,” arXiv: 1707.03429, 2017.

[12] M. Hayashi, K. Iwama, H. Nishimura, R. Raymond and S. Yamashita, “(4, 1)-quantum random access coding does not exist,” *New Journal of Physics*, **8**(8), p. 129, 2006.

[13] K. Iwama, H. Nishimura, R. Raymond, and Shigeru Yamashita, “Unbounded-error one-way classical and quantum communication complexity,” *International Colloquium on Automata, Languages, and Programming*, pp. 110–121, 2007.

[14] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik and J. L. O’Brien, “A variational eigenvalue solver on a photonic quantum processor,” *Nature Communications*, **5**, article number: 4213, 2014.

