jh240049

NISQ 時代を見据えたバッチ型量子回路シミュレータの開発

高橋慧智 (大阪大学 D3 センター)

概要

本研究では，Noisy Intermediate-Scale Quantum Computer (NISQ) 時代における量子計算アルゴリズムの開発・評価に資するバッチ型量子回路シミュレータを開発する．ノイズを有する量子計算機のシミュレーションや変分量子アルゴリズムの評価を効率的に行うためには，多数の量子回路の状態を効率的に計算できる必要があるが，既存のシミュレータの多くではバッチ型計算の機能が十分ではない．本年度は，東北大 AOBA-S に搭載されている Vector Engine Type プロセッサを実装対象とし，(1) バッチ型計算に適する並列化・ベクトル化方式の検討，(2) 多数の量子回路を効率的に表現するための API の設計，(3) NISQ 向け量子アルゴリズムを用いた性能評価，の 3 点の研究項目に取り組み，Python インターフェースを備えるバッチ型量子回路ライブラリ veqsim を設計・実装した．

1 共同研究に関する情報

1.1 共同研究を実施した拠点名

- 東北大学 サイバーサイエンスセンター

1.2 課題分野

- 大規模計算科学課題分野

1.3 参加研究者の役割分担

- 高橋慧智 (大阪大学 D3 センター): 全体統括，コード開発，性能分析
- 森俊夫 (大阪大学 量子情報・量子生命研究センター): コード開発，性能評価

2 研究の目的と意義

本研究では，ノイズ耐性量子アルゴリズムや変分量子アルゴリズムの研究開発に資する，多数の量子回路を同時にシミュレート可能なバッチ型量子回路シミュレータの開発を目的とする．量子計算に注目が集まるにともない，様々な量子回路シミュレータが開発されている．ただし，これらのシミュレータのほとんどはノイズが存在しない理想的な量子計算機を再現するものである．一方，量子計算機の開発は依然として萌芽期にあるのが実情であり，短期的にはノイズが大きく量子ビット数も限定的な Noisy Intermediate-Scale Quantum (NISQ) 時代が続くと考えられている．

本研究では，NISQ 量子計算機を対象とした量子アルゴリズムの開発には，次の理由から多数の量子回路を効率的にシミュレートするバッチ型シミュレータが有用であると考える．1 点目は，ノイズを有する量子計算機のシミュレーションである．NISQ 量子計算機を実用化するためには，量子誤り訂正などノイズの影

響を抑制するための手法の開発が不可欠である．モンテカルロ法に基づくノイズシミュレーション手法では，それぞれ異なるノイズを注入した多数の量子回路をシミュレートし，それらのアンサンブルにより量子状態を推定する．このような手法の効果的な実装には，効率的に多数の回路をシミュレーションすることが必須である．2 点目は，変分量子アルゴリズム (Variational Quantum Algorithm, VQA) の評価である．一般に VQA では，(1) 量子計算機上でパラメータ化された量子回路を実行し，(2) その出力をサンプリングし，(3) 古典計算機上で損失関数が最小になるようにパラメータを更新する，という手順を繰り返す．このようなアルゴリズムを評価するためには，パラメータが異なる多数の量子回路をシミュレートする必要があり，やはりバッチ型の量子回路シミュレータが必要である．以上の理由から，NISQ 時代を見据えた量子回路シミュレータには効率的なバッチ型シミュレーション機能が必要であると考え，開発に取り組む．

本共同研究は，高性能計算の専門家である課題代表者と量子ソフトウェアの専門家である課題副代表者が共同で推進する学際的な課題である．課題副代表者が所属する大阪大学量子情報・量子生命センター (Quantum Information and Quantum Biology Center, QIQB) は国産の量子計算機に加え，国内で広く使用されている量子回路シミュレータ Qulacs[*1] を独自開発しており，課題副代表者の森は Qulacs の開発を主導する立場にある．そのため，QIQB 内外の量子アルゴリズムの研究者と常に対話し，ニーズを把握している．本課題が目指すバッチ型シミュレータもそのようなユーザとの対話から着想を得たものである．それゆえ，単に高性能計算の観点から量子回路シミュレータの性能を追求するだけでなく，実際に量子計算の専門家が必要としている機能を実装できる．また，現在 Qulacs はコードを刷新した新バージョンの開発に着手しており，本研究で得られた知見や開発したソースコードを取り入れることにより，本研究成果の幅広い利活用を目指す．

3 当拠点公募型研究として実施した意義

状態ベクトル型の量子回路シミュレーションは，量子ビット数を n として 2^n 要素の複素ベクトルに対して量子ゲートに対応する疎な行列 (ゲート行列) を乗算することにより実現されるため，メモリ律速であることが広く知られている．東北大学サイバーサイエンスセンターが有するスーパコンピュータ AOBA-S は NEC 社製 SX-Aurora TSUBASA システムを採用しており，メモリアクセス性能において CPU を大きく凌駕し，GPU と同等以上の性能を発揮する．そのため，AOBA-S は本課題で開発するシミュレータの実行基盤として最適な計算資源である．

4 前年度までに得られた研究成果の概要

該当なし

5 今年度の研究成果の詳細

今年度の研究では，Vector Engine 上で動作し，多数の量子回路の状態を効率的に計算可能なバッチ型量子回路シミュレータ veqsim を設計，実装，および評価した．開発したシミュレータのソースコードは，GitHub 上で MIT ライセンスの下で公開している[*2]．以下では

[*1] https://github.com/qulacs/qulacs

[*2] https://github.com/keichi/veqsim

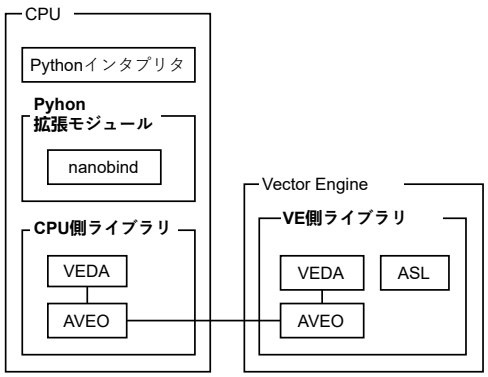


図 1: veqsim のアーキテクチャ

veqsim の開発に係る個別の成果について記す．

5.1 veqsim の基本設計

量子情報分野におけるシミュレータの利用実態を調査したところ，実装の容易さやライブラリの充実度から Python 言語の API が広く用いられていることが明らかになった．そこで，本研究において開発するシミュレータも Python 言語に対応させる必要があると判断した．Vector Engine は C/C++ または Fortran で記述されたプログラムを実行可能であるものの，Python インタプリタを動作させた実績はない．実際に CPython 3.13.3[*3]を Vector Engine 向けにクロスコンパイルを試みたところ，多数の文法エラーが発生した．加えて CPython が依存する libffi や libopenssl などのライブラリも Vector Engine での動作実績が存在しないことから，Vector Engine 上で Python インタプリタを実行することは困難と判断した．

そこで，Python インタプリタはホスト上で実行し，カーネルのみ Vector Engine にオフロードする設計を採用した．これは PyTorch や CuPy などの GPU を活用する Python ライブラリにおいても採用されている方式である．Vector Engine では，通常 Vector Engine 上でプログラム本体が実行され，システムコールのみがホスト上で実行される．そのため，GPU などのアクセラレータのように，プログラム本体をホスト上で実行し，一部の処理のみを Vector Engine にオフロード可能にするをフレームワークである Alternative Vector Engine Offloading (AVEO)[*4] を活用した．さらに，AVEO をラップし，CUDA Driver API に類似した API を提供する VE Driver API (VEDA)[*5] を採用した．これは，将来的に GPU への対応をも見据え，コードの再利用可能性を高めるためである．

図 1 に veqsim のアーキテクチャの概要を示す．veqsim は主に 3 つのコンポーネントから構成した．すなわち, (1) Python 拡張モジュール，(2) CPU 側ライブラリ，(3) VE 側ライブラリ，である．Python 拡張モジュールの実体は x64 用共有ライブラリであり，Python インタプリタによってロードされ，CPU 側ライブラリへのインターフェースを Python プログラムへ提供する．Python インターフェースは nanobind[*6]と呼ばれる C++ ライブラリの Python バインディングを作成するためのライブラリを用いた．類似した目的のライブラリは他にも存在するが，他ライブラリと比較すると nanobind は生成されるバイナリサイズおよび実行時オーバーヘッドが小さいことから採用した．

CPU 側ライブラリは C++ API を提供する x64 共有ライブラリであり，Python 拡張モ

[*3] https://www.python.org/downloads/release/python-3133/

[*4] https://github.com/SX-Aurora/aveo

[*5] https://github.com/SX-Aurora/veda

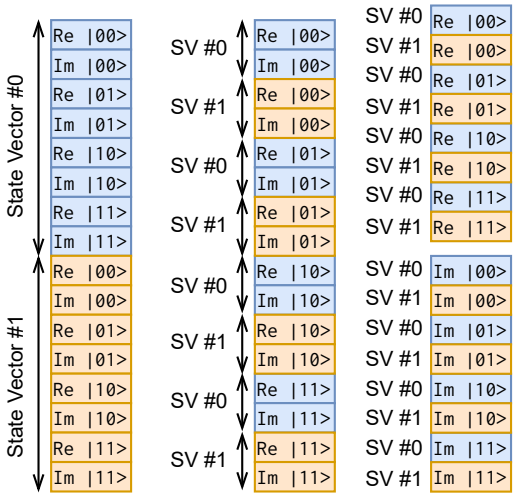
[*6] https://github.com/wjakob/nanobind

ジュールによってリンクされる．CPU 側ライブラリは VEDA を用いて Vector Engine を制御し，各ゲートの定義に従って量子状態を更新するなど，主要な計算処理を VE にオフロードするとともに，状態ベクトルなどのデータホスト・VE 間で転送する役割を担う．VE 側ライブラリは各ゲートに対応する処理など実際の計算処理を実装しており，VE 向けの共有ライブラリである．これは VEDA によってロードされ，CPU 側ライブラリからの要求に応じて実行される．

5.2 バッチ型計算に適する並列化方式の検討

単一の量子回路のシミュレーションでは状態ベクトルに対する疎行列ベクトル積を並列化するが，多数の量子回路を同時にシミュレーションする場合は，さらに回路レベルでの並列性も利用可能である．そこで，本研究において実装対象とする Vector Engine に適した状態ベクトルのメモリ配置，および，状態の更新処理の並列化方式を比較評価した．

図 2 に本研究において検討した 3 種の状態ベクトルのメモリ配置方法を示す．図 2(a) は NVIDIA 社が提供する状態ベクトル型量子回路シミュレーションライブラリである cuStateVec[*7] で使用されているメモリ配置であり，各回路の状態ベクトルが連続してメモリ上に配置される．図 2(b) は異なる回路の同じ基底の成分がメモリ上で連続になる配置である．図 2(c) のメモリ配置は図 2(b) に類似しているが，状態ベクトルの実部と虚部を別々の配列に格納する方式である．これらそれぞれのメモリ配置を用いた際の 1 量子ビットゲートの実行時間を評価した結果，(c) のメモリ配置は (b) および (a) のメモリ配置に比べそれぞれ最大で 6.9 倍および 1.9 倍高速であることが明らかになった．したがって，veqsim では (c) のメモリ配置を採用することに決定した．



(a) AoS 配置) (b) SoA 配置 A (c) SoA 配置 B

図 2: 状態ベクトルのメモリ配置 (バッチサイズ=2，量子ビット数=2 の例)

Vector Engine ではベクトル化とスレッド (もしくはプロセス) 並列化の 2 段階の並列化が可能である．veqsim では，図 2(c) のメモリ配置に従って状態ベクトルの同一基底成分をメモリ上に連続に配置した上で，異なる回路についてベクトル化し，行列ベクトル積をスレッド並列化することとした．これは，行列ベクトル積をベクトル化すると状態ベクトルへのメモリアクセスがストライドアクセスになる場合があり，実効メモリ帯域幅が低下するためである．一方，採用した並列化方式では常に連続メモリアクセスを実現できることから，高い実効メモリ待機幅を実現できると期待できる．

5.3 多数の量子回路表現する API の設計

veqsim において，ユーザに対して提供する Python API を検討した．最も柔軟な API は，任意の複数の回路の集合を与えるとそれらの状態を返却するようなものである．しかし，実際

[*7] https://docs.nvidia.com/cuda/cuquantum/latest/custatevec/index.html

のユースケースを調査すると，全く異なる構造の回路を多数シミュレートしたいという要求は少なく，同一もしくは類似した回路の一部のパラメータ (回転ゲートの回転角度など) を変えて多数の回路をシミュレートしたいという要求が多いことがわかった．そこで，量子状態ベクトルのバッチに対して，一括してゲート操作を適用する API を提供する設計を採用した．

ソースコード 1 に示す veqsim を用いた量子回路シミュレーションの記述例に基づいて，具体的な API を説明する．API の中核を成すのは，量子状態を保持する `State` というクラスである．このクラスは同一量子ビット数の多数の量子状態を保持する．`State` は各量子ゲートに対応するメソッドを備える．例えば，`act_x_gate()` であれば，バッチ内の全ての量子状態に対して，X ゲートを作用させる．回転ゲートなどのパラメータを有する量子ゲート，例えば RX ゲートはオーバーロードされており，バッチ内の全ての量子状態に対して，(1) 同じゲートを作用されるものと，(2) 量子状態ごとに異なるゲートを作用させるものがある．具体的には，RX ゲートに対応するメソッドは `State.act_rx_gate(theta, target)` であり，`theta` がスカラである場合は全ての量子状態に対して同一の回転角度 θ の RX ゲートぃ RX(θ) が適用される．一方，`theta` がベクトルである場合は，i 番目の量子状態に対して，回転角度 θ_i の RX ゲート RX(θ_i) が適用される．

ソースコード 1: veqsim を用いた量子回路シミュレーションの記述例

```
from veqsim import State
state = State(qubits=8, batch_size=100)
state.set_zero_state()
state.act_rx_gate(phi[0], 0)
state.act_rz_gate(phi[1], 0)
state.act_rx_gate(phi[2], 1)
state.act_rz_gate(phi[3], 1)
state.act_cx_gate(0, 1)
state.act_rz_gate(phi[4], 1)
state.act_rx_gate(phi[5], 1)
print(state.get_vector(0))
```

ノイズゲートは特殊なゲートであり，作用させる量子状態とゲートを確率的に決定する．veqsim では Depolarizing ノイズと呼ばれる標準的なノイズモデルを実装している．これは，エラー確率 $0 \leq \lambda \leq 1$ が与えられたとき，確率 $\frac{\lambda}{3}$ で対象の量子ビットに対して X, Y，Z のいずれかの Pauli ゲートを作用させ，$1-\lambda$ で何も作用させないというものである．veqsim では，バッチ内の各量子回路に対して確率的に以上の操作を行う．

5.4 NISQ 向け量子アルゴリズムを用いた性能評価

予備的な研究において単一ゲートレベルの動作確認と基礎的な性能評価は完了しているが，実際に研究されている NISQ 向け量子アルゴリズムを用いた性能評価が必要である．そこで，まず，様々な NISQ 向けアルゴリズムの基礎となる変分量子固有値ソルバー (Variational Quantum Eigensolver, VQE) と呼ばれるアルゴリズムを本シミュレータ上に実装した．

VQE は変分法に基づきハミルトニアンの基底状態を求めるためのアルゴリズムである．変分原理とは，式 (1) に示す通り，任意の状態 $|\psi\rangle$ についてそのエネルギー期待値が基底エネルギー E_0 より高くなることを示す．

$$\langle\psi\,|\,H\,|\,\psi\rangle \geq E_0 \tag{1}$$

この変分原理の応用が変分法であり，様々な状態 $|\psi\rangle$ を生成し，それらの期待値の最小値によって基底エネルギーを近似する．VQE は変分法を量子計算機で効率的に実行可能に実装したものであり，全体像は次のとおりである:

1. パラメータ θ によってパラメータ化された

量子回路を用い，量子状態 $|\psi(\theta)\rangle$ を生成する．

2. エネルギー期待値 $\langle\psi(\theta)\,|\,H\,|\,\psi(\theta)\rangle$ を測定する．
3. 古典計算機で最適化アルゴリズムによってエネルギーが小さくなるような θ を決定する．
4. 1. に戻る．

VQE の実装にあたっては，量子状態 $|\psi\rangle$ を構築するための各種量子ゲートに加えて，量子状態について与えられたハミルトニアンの期待値を計算する機能が必要となる．この機能を実装し，水素化ヘリウムイオンの基底エネルギーを求める VQE 回路を実装した (ソースコード 2).

ソースコード 2: VQE による水素化ヘリウムイオンの基底エネルギーの探索

```
def cost(theta):
  obs = Observable()
  obs.add_operator(
    PauliOp(-3.8505 / 2, [0, 1], [0, 0])
  )
  ...

  noise_rate = 0.001

  state = State(2, 10000)
  state.set_zero_state()
  state.act_rx_gate(theta[0], 0)
  state.act_depolarizing_gate_1q(
     0, noise_rate
  )
  ...

  energy = np.mean(state.observe(obs))

  return energy

init = np.random.rand(6)
res = scipy.optimize.minimize(
    cost, init, method="Powell",
    options={"maxiter": 100},
    callback=callback
)
```

veqsim で VQE 回路を実行したところ，高精度で厳密解に収束することを確認できた．また，回路中に Depolarizing ノイズゲートを挿入し，ノイズ発生確率を増加させると，得られた基底エネルギーの近似値が厳密解から少しずつ離れることも確認できた．以上のことから，実際の NISQ 向けアルゴリズムを veqsim によって評価可能であることを実証できた．

6　今年度の進捗状況と今後の展望

申請時に設定した研究項目 (1) バッチ型計算に適する並列化・ベクトル化方式の検討, (2) 多数の量子回路を効率的に表現するための API の設計，(3) NISQ 向け量子アルゴリズムを用いた性能評価，のうち，(1) および (2) については 100% 達成できたと自己評価する．(3) については，重要な NISQ 向けアルゴリズムである VQE の実装と評価を実現できたが，当初予定していた量子サポートベクトルマシンや量子ニューラルネットワークなどの量子機械学習アルゴリズムを用いた評価までは達成できなかった．よって，(3) は 60% 達成できたと自己評価する．

本年度に量子回路シミュレータとして核となる機能は実装が完了したと考えている．今後は，種々の量子機械学習アルゴリズムや量子近似最適化アルゴリズム (Quantum Approximate Optimazation Algorithm, QAOA) など，さらに複雑な NISQ 向けアルゴリズムを実装できるか検証を行い，追加が必要な機能があれば実装していく．また，本年度はコードの開発に集中したため，論文発表成果がない．そのため，来年度以降は論文発表に取り組む予定である．