Vol.2026-HPC-203 No.58
Vol.2026-QS-17 No.58
2026/3/18

FTQCにおける表面符号の物理量子ビットへのノイズ適応型配置の性能評価

丸茂 直樹[1,a)]　木村 啓二[1,b)]

概要：大規模な量子計算を行うために誤り耐性量子コンピュータ（FTQC）が注目される．FTQC 向けの符号の中でも，表面符号はハードウェアへの要求性能が低いことなどから有望な符号の 1 つである．これまでの表面符号の研究の多くは符号の構成等を工夫し，ハードウェアへの要求性能を低くすることを目的としており，物理エラーを一様であると仮定していた．しかし，実際のところ実機の物理エラー率は非一様である．そこで，本稿では非一様な物理エラー環境における論理エラー率の低減手法として，キャリブレーションデータを用いたノイズ適応型配置を提案する．ノイズ適応型配置とランダム配置の論理エラー率をシミュレーションにより評価した結果，ノイズ適応型配置ではランダム配置の論理エラー率に対して最大で約 21%の低減効果が確認できた．また，符号距離が 3 の場合にはノイズ適応型配置を行うと，最大 92%の確率でランダム配置よりも低い論理エラー率となった．さらに，マッピングによる論理エラー率低減の上限と比較した際に，本稿で定義したノイズ適応型配置モデルの最適化率は最大で約 59%に達した．

キーワード：FTQC，表面符号，ノイズ適応型配置

Performance evaluation of noise-aware mapping of surface code to physical qubits in FTQC

Naoki Marumo[1,a)]　Keiji Kimura[1,b)]

Abstract: Fault-Tolerant Quantum Computing (FTQC) is attracting attention as a means of enabling large-scale quantum computing. The surface code is a promising code for FTQC since it does not require high hardware performance. Much of the previous research aimed to reduce the hardware requirements by optimizing the code composition. Therefore, they assume a uniform physical error rate across all Qubits, though this is overly simplistic. In this article, we propose noise-aware mapping for the surface code using calibration data to reduce the logical error rate in a non-uniform physical environment. As a result of simulation, noise-aware mapping reduced about 21% in maximum in the view of the rate of its logical error against that of random mapping. When code distance was 3, the logical error rate of noise-aware mapping was lower than that of random mapping in 92%. In addition, compared with the upper limit of optimization, the proposed noise-aware mapping model achieved about 59% in maximum.

Keywords: FTQC, Surface code, Noise-aware mapping

1. はじめに

量子ゲート方式の量子コンピュータは機械学習や化学計算などの様々な用途で利用可能であり，古典コンピュータでは困難な問題への応用が期待されている．しかし，その計算過程における時間経過や，量子ビット間の干渉によりエラーが生じる．このようなエラーは計算規模に応じて影響も大きくなる．量子コンピュータが古典コンピュータに対して優位性を持つ大規模な量子計算において，理

[1] 早稲田大学
Waseda University, Shinjuku, Tokyo 169–8555, Japan
[a)] naoki_marumo@fuji.waseda.jp
[b)] keiji@waseda.jp

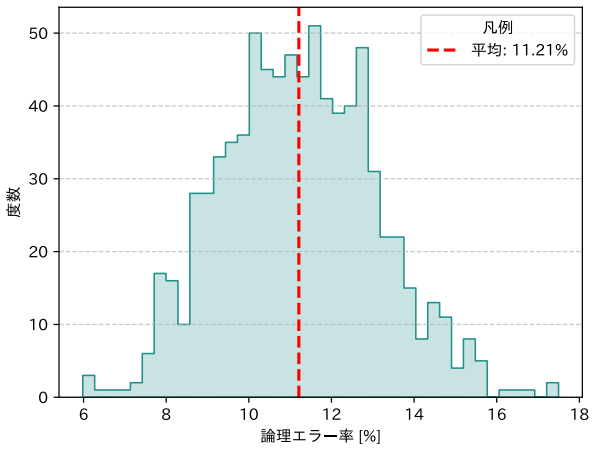

図 1 マッピングの違いによる論理エラー率の変動
（符号距離 3，シンドローム測定サイクル 25，初期状態 $|0\rangle$）

想的な結果を得るための手法として量子誤り訂正（QEC）[1] が提案されている．そして QEC を行うための大規模な量子ゲート方式の量子コンピュータを FTQC[2] という．FTQC では計算過程において誤り訂正を行うことにより，計算結果の信頼性を保証する．

様々な FTQC 向けの符号があるが，なかでも表面符号 [3], [4] はハードウェアへの要求性能の低さや構造の単純さから有望視されている．従来の研究は符号化の方法や復号方法の改善による論理エラー率の低減を提案するものが多い．これらの先行研究では物理量子ビットのエラー率が一様であることを前提としている．

なかには非一様な物理エラーを前提として，論理エラー率の低減を目的とする研究もある．具体的には，エラー率が非一様である時の論理エラーを防ぐためのマッピング手法を提案する研究 [5] や，物理的なパルスレベルでの最適化を必要とする研究 [6], [7] などが挙げられる．しかしながら，これらの先行研究の提案をユーザが実用的に利用することは困難である．

前述のように先行研究の多くでは一様な物理エラーを前提とするが，実機の物理量子ビットはそれぞれ異なるエラー率を持つ．物理量子ビットのエラー率が非一様であることを前提とすると，論理エラー率の度数分布は図 1 のようになる．この図から，同じ符号を構成する場合でもマッピング箇所が異なることにより，論理エラー率が 6%から 17%の間で変動することが確認できる．すなわちマッピングの工夫によって論理エラー率を低減できることを示唆している．なお，図 1 のノイズモデルは 4 章，実験条件は 5.2.1 節にて述べる．

現在実現されているゲート方式の量子コンピュータである Noisy Intermediate-Scale Quantum（NISQ）でも物理エラーは非一様である．そこで NISQ では，高精度な計算結果を得るために，キャリブレーションデータを使用したノイズ適応型配置が実用化されている．この手法で使用するキャリブレーションデータはユーザからもアクセスが容易であり，最適化も目的関数の最大化で定義できるため，現在の NISQ トランスパイラの中で実用化されている．

本稿では，非一様な物理エラーを前提として，物理量子ビットのエラー率などのキャリブレーションデータを用いたノイズ適応型配置を提案し，その表面符号のマッピングにおける効果を検証した．

2. FTQC

ゲート方式の量子コンピュータでは，時間経過によるエネルギー緩和や位相緩和，物理量子ビット間のクロストーク，不正確な制御パルスによるゲート操作により，計算過程でエラーが発生する．FTQC では複数の物理量子ビットを使用し，1 つの論理量子ビットを構成することにより，情報を冗長化する [1], [8]．そして計算過程において誤りの検出，訂正を行う．これにより，大規模な量子計算におけるエラーを低減する．情報の冗長化を行い，1 つの誤り訂正符号を構成するためには，大量の物理量子ビットが必要となる [4]．また，符号化に伴うゲート操作も必要となる．

FTQC では情報の冗長化による誤り訂正を実現する一方で，使用する物理量子ビットやゲート操作の増加に伴いエラーの原因も増加する．そのため，物理量子ビットにおけるエラー率が閾値よりも大きい場合，誤り訂正の効果よりも冗長化に伴う物理エラーの増加の影響の方が大きくなる．逆に，物理エラー率が閾値未満であれば，符号距離を大きくし，冗長性を高めることにより，論理量子ビットのエラー率を任意の値まで低減することができる [9]．これを閾値定理という．

従来の FTQC の研究は閾値定理を念頭に置いて，符号化方式の改善により，ハードウェアへの要求性能である閾値を大きくすることを目標とすることが多い．この場合，符号の性能評価が目的であるため，各物理量子ビットのエラー率は一様であると仮定する．しかしながら，実際の物理エラー率は物理量子ビットごとに異なる．そのため，同じ符号を構成するとしても，その符号の物理量子ビットへのマッピング次第で論理量子ビットのエラー率は変動すると考えられる．

3. 表面符号

FTQC の符号化方式として，Shor 符号 [1]，Steane 符号 [8]，表面符号 [3], [4]，量子 LDPC 符号 [10] などが提案されている．なかでも表面符号は 1 ゲート操作におけるエラーの許容度が約 1%と比較的大きい [11]．また，誤りの検出，訂正を行うためのスタビライザが近傍の物理量子ビットのみで構成できる．さらに，論理ゲート操作の実装において物理量子ビットが全結合であることを必要としない．このように，表面符号では，物理量子ビットに求められる接続やエラー率などの条件が緩い．また，2 次元の格子と

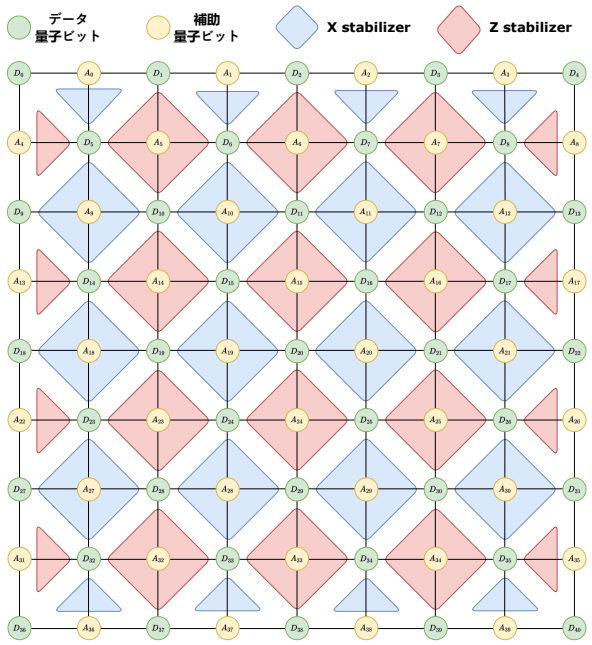

図 2 符号距離 5 の表面符号の構成

いう符号構成の単純さから拡張性が高く，符号距離を大きくすることが容易である．このようなハードウェアへの要求性能や拡張性などの観点から，表面符号は FTQC の符号として有望視されている．

図 2 に符号距離 5 の表面符号の構成を示す．各データ量子ビットは情報を持ち，その間の補助量子ビットは誤り検出のためのシンドローム測定に使用する．1 つの補助量子ビットとそれを囲む 2〜4 個のデータ量子ビットでスタビライザが構成される．スタビライザには Z エラー（位相反転）を検出する X スタビライザと X エラー（ビット反転）を検出する Z スタビライザがある．スタビライザはその固有値が $+1$ となるように構成される．そして各スタビライザに対して反交換なエラーが発生したときに固有値（シンドローム）が -1 に変化することを用いて誤りを検出する．

表面符号での誤り訂正は 1 回のシンドローム測定ごとに行われるのではなく，数回分の時間経過に応じた連続的な誤り検出のデータをもとに行われる．エラーが発生するとそのエラー鎖の両端のシンドロームが -1 となる．しかしこれだけでは誤りの発生したデータ量子ビットを特定することはできない．そこで，Minimum Weight Perfect Matching（MWPM）を行う [4]．単純な復号では，すべてのエッジの重みが等しい MWPM によって，シンドロームが -1 となっている補助量子ビット間の距離が最小となる経路を見つけ，その経路上のデータ量子ビットで誤りが発生したものと認識する．物理量子ビットのエラー率が一様である場合にはこの手法が有効である．しかし，実機の物理量子ビットのエラー率は非一様である．そこで，エッジの重みを非一様な物理エラー率に合わせて定義する手法をとると復号性能が向上する [5]．現在のシミュレーションツールには物理エラー率に応じた重み付きのデコーダが実装されている．

論理量子ビットでの論理 X ゲートは Z スタビライザの境界間のすべてのデータ量子ビットに物理 X ゲートをかけることで実現できる．例えば，境界間のすべてのデータ量子ビットにおいて X エラーが発生すると論理 X ゲートをかけた時と同じ状態になる．このような場合に論理エラーが発生する．これは論理 Z ゲートでも同じことがいえる．また，表面符号において符号距離が d の時，誤り検出可能なデータ量子ビットは $d-1$，誤り訂正可能なデータ量子ビットは $(d-1)/2$ と限界が定まっている [4]．

4. ノイズモデルの定義と回路への挿入

4.1 シミュレーションにおけるノイズモデル

本稿では 156 物理量子ビットを実装する IBM 社の NISQ である IBM Fez のキャリブレーションデータを使用した．キャリブレーションデータの外れ値を除き，取得した平均と標準偏差をもとに，最小と最大に収まる切断正規分布となるようにランダムな値でエラー率とコヒーレンス時間を定義した．すなわち，分布は実機と同様で，値はランダムとなるようにノイズモデルを作った．以下に実機から取得するパラメータを挙げる．

- 単一量子ビットゲート操作の duration time
- 複数量子ビットゲート操作の duration time
- 測定の duration time
- リセットゲート操作の duration time
- T_1，エネルギー緩和時間
- T_2，位相緩和時間
- X ゲート操作エラーの確率
- CZ ゲート操作エラーの確率
- 測定操作エラーの確率

エラー率やコヒーレンス時間などの性能は日々変化するので，複数回分これらのパラメータを取得する．なお，ゲートエラーとして示されるエラーは，あらゆる原因を含んだゲート操作によって生じるエラーとパルスの不正確さや製造上の不完全さに起因するエラーの，いずれも指すことがあるが，本稿では前者をゲート操作エラー，後者を潜在的操作エラーと述べる．

キャリブレーションデータにはリセットゲートのゲート操作エラーが含まれていない．そこで，duration time が近い測定操作エラーにて代替した．正確なノイズモデルの構築にはゲート操作エラーをデコヒーレンスエラーと潜在的操作エラーに分けてモデル化する必要もある．本稿ではデコヒーレンスエラーのエラー率 e_t を，各操作の duration time t，緩和時間 T_1, T_2 を用いて，式 1 で定義した [12], [13].

$$e_t(t) = \frac{3 - e^{-\frac{t}{T_1}} - 2e^{-\frac{t}{T_2}}}{6} \tag{1}$$

表 1 ノイズモデルの指標とランダム値生成のための入力値の例

指標		平均	標準偏差
エネルギー緩和時間 T_1 [μs]		156.47	45.67
位相緩和時間 T_2 [μs]		130.15	44.49
潜在的操作エラー率 [%]	単一量子ビットゲート	0.0172	0.0092
	複数量子ビットゲート	0.2398	0.0813
	測定	0.5964	0.6284
	リセットゲート	0.5964	0.6284

表 2 操作に応じたエラーの種類とタイミング

操作の種類	エラーの種類	タイミング
ゲート操作待ち	DEPOLARIZE1	操作後
単一量子ビットゲート	DEPOLARIZE1	操作後
複数量子ビットゲート	DEPOLARIZE2	操作後
測定	X_ERROR, Z_ERROR	操作前
リセットゲート	X_ERROR, Z_ERROR	操作後

そして，潜在的操作エラー e_g をゲート操作エラー e_{all}，デコヒーレンスエラー e_t を用いて式 2 で定義した．

$$e_g = e_{all} - e_t \tag{2}$$

取得したキャリブレーションデータと定義したエラーにより，ノイズモデルを構成する．ノイズモデルの指標とその平均，標準偏差の一例を表 1 に示す．本稿ではノイズモデルにおいて潜在的操作エラーとデコヒーレンスエラーを考慮する．

4.2 回路へのノイズの挿入

本稿の評価で使用した Stim では，表面符号の論理量子メモリの性能評価用回路を生成することができる．しかし，非一様な物理エラーを前提とする場合，生成した回路操作に応じて，エラーを後から挿入する必要がある．表 2 に，操作に応じたエラーの種類とその挿入タイミングを示す．なお，測定とリセットゲートにおけるエラーの種類は操作の基底に応じて決まる．

5. FTQC のノイズ適応型配置モデルの定義

5.1 マッピングの指標

ノイズ適応型配置にはマッピング箇所を評価する指標が必要となる．NISQ におけるノイズ適応型配置では，計算結果への影響が大きい複数量子ビットゲート操作の信頼性と測定操作の信頼性をマッピング箇所の評価指標としていた [14]．しかし，FTQC の論理エラー率への影響が大きい指標はまだわかっていない．そこで本稿では，ゲート操作やエラーの種類に応じた信頼性と iOlius 等の提案手法 [5] を指標の候補とした．

マッピング箇所の信頼性は各ゲートの信頼性の積で計算可能である（式 3）．

$$R_{Map} = \prod_{q_i \in Mapping} (1 - e_{q_i}) \tag{3}$$

式 3 は信頼性なので大きいほどよい．しかし，本稿では目的関数を指標の線形結合の最小化問題とするほか，符号距離の変化に伴う信頼性の変化に対応するために，エラーを指標とし，式 4 で定義した．

$$f_R = -\frac{\log R_{Map}}{\#Qubits_{\in Map}} \tag{4}$$

iOlius 等は表面符号において，物理エラー率が高いデータ量子ビットの周囲を物理エラー率が低いデータ量子ビットで囲むことにより，論理エラー率を低減する手法を提案した [5]．本稿では，この手法を関数として定義する．iOlius 等はエラーの原因として，デコヒーレンスエラーのみを考慮している．まず，実装されている物理量子ビットすべての平均の緩和時間（$\bar{T_1}$，$\bar{T_2}$）を式 1 に代入し，ある時間 t における平均のデコヒーレンスエラーの確率 $\bar{e_t}(t)$ を定義する．そして，平均のデコヒーレンスエラーの確率 $\bar{e_t}(t)$ が $(1-e^{-1})/2$ となる時間 t_τ を解析的に求める．各データ量子ビットの性能 $e_t(\tau)$ を，その緩和時間（T_1，T_2），継続時間 t_τ を式 1 に代入して求める．そして各データ量子ビットの性能偏差を式 5 により求める．

$$\Delta e_t = e_t(\tau) - \bar{e_t}(\tau) \tag{5}$$

最終的には隣り合うデータ量子ビット間の性能偏差の積をマッピング全体で足し合わせ，それをデータ量子ビット間のエッジ数で割った値を，iOlius 等の提案手法によるマッピング評価指標 f_{map} とする（6）．ここで D は隣接するデータ量子ビットのペアの集合を表す．

$$f_{map} = \frac{1}{|D|} \sum_{q_i, q_j \in D} (\Delta e_{t,q_i} \times \Delta e_{t,q_j}) \tag{6}$$

最終的に使用した 9 種類のマッピング指標の候補を以下に挙げる．いずれの指標も小さいほど性能が良いことを示すように定義した．

- 単一量子ビットゲートのゲート操作エラー
- 複数量子ビットゲートのゲート操作エラー
- 測定のゲート操作エラー
- リセットゲートのゲート操作エラー
- 単一量子ビットゲートのデコヒーレンスエラー
- 複数量子ビットゲートのデコヒーレンスエラー
- 測定のデコヒーレンスエラー
- リセットゲートのデコヒーレンスエラー
- iOlius 等の提案手法（エッジ毎の性能偏差の積の和）

5.2 各マッピングの指標候補と論理エラー率の関係性評価

5.2.1 評価環境

表面符号の論理量子メモリ性能に影響を与え，ノイズ適応型配置の目的関数に組み込むべきマッピングの評価指標を決めるため，各マッピングの評価指標候補と論理エラー率の関係性を調べる．本稿の評価では誤り訂正回路のシ

表 3　各マッピング指標候補と論理エラー率の相関係数

単一量子ビットゲートのゲート操作中のエラー	-0.0257
複数量子ビットゲートのゲート操作中のエラー	0.0622
測定操作中のエラー	0.1886
リセットゲートのゲート操作中のエラー	0.0771
単一量子ビットゲートのデコヒーレンスエラー	0.3465
複数量子ビットゲートのデコヒーレンスエラー	0.3346
測定のデコヒーレンスエラー	0.3467
リセットゲートのデコヒーレンスエラー	0.3467
iOlius らの提案手法	0.1867

ミュレーションツールである Stim を使用した．評価の条件は以下に述べる．本評価では，4 章で述べた方法により，3 回分の実機のキャリブレーションデータからそれぞれ 10 種類のランダムなノイズモデルを生成し，使用した．

物理量子ビット数　169（13 × 13 の格子）

符号距離　3，5

シンドローム測定サイクル　符号距離数（3 または 5），25

データ量子ビットの初期状態　$|0\rangle$，$|+\rangle$

#shots　1000000

上の条件において，各マッピングの評価指標候補と論理エラー率について，相関係数の観点から関係性を評価した．

5.2.2　評価結果

符号距離やシンドローム測定サイクル，初期状態の条件ごとに相関係数について評価した．表 3 に符号距離が 3，シンドローム測定サイクルが 25，初期状態が $|0\rangle$ の時の各マッピングの評価指標候補と論理エラー率の相関係数を示す．

表 3 から，論理エラー率との相関が最も強いマッピング指標の候補は，相関係数が 0.3467 である測定のデコヒーレンスエラーとリセットゲートのデコヒーレンスエラーだとわかる．そして相関の強い指標は単一量子ビットゲートのデコヒーレンスエラー，複数量子ビットゲートのデコヒーレンスエラーと続く．デコヒーレンスエラーの相関係数はゲートによらず 0.3 を超えており，ゲート操作エラー等の指標よりも相関が強い．デコヒーレンスエラーの原因はいずれの場合も緩和時間（T_1，T_2）に依存している．すなわち，論理エラー率と緩和時間の間には関係性があることがわかる．

4 種類のデコヒーレンスエラーの次に相関が強いのは相関係数が 0.1886 である測定操作中のエラーだった．測定は他のゲート操作と比較すると，duration time が大きく，物理エラー率も大きいことから論理エラー率への影響も大きくなる．

そして，相関係数が 0.1867 である iOlius 等の提案手法がその次に相関が強かった．これはマッピング中の物理量子ビットのエラー率だけではなく，そのエラー率の分布も論理エラー率に影響を与えることを示唆する．

他の 3 種類のゲート操作中のエラーはいずれも相関係数の絶対値が 0.1 未満であり，ほとんど相関はなかった．

符号距離，シンドローム測定サイクル，初期状態を変えても，各マッピング指標の候補と論理エラー率の相関係数は各ゲート操作のデコヒーレンスエラー，測定ゲート操作中のエラー，iOlius 等の提案手法において比較的大きくなった．また，これらの関係性はそれぞれのマッピング指標の候補と論理エラー率の関係性を図で確認した結果からも同様の結論が導出できた．

5.3　ノイズ適応型配置の目的関数の定義

NISQ におけるノイズ適応型配置と同様に，結果に影響を与えるマッピング指標の絞り込みを行う．5.2.2 節より，表面符号における論理エラー率と相関が強いマッピング指標の候補が各ゲート操作のデコヒーレンスエラー，測定ゲート操作中のエラー，iOlius 等の提案手法の 6 種類とわかった．4 種類のゲート操作のデコヒーレンスエラーは，すべて原因が緩和時間（T_1，T_2）であることから，その中の 1 種類の指標のみを残せば十分に論理エラー率への影響を評価できる．そこで，本稿では最も相関係数の大きかった測定のデコヒーレンスエラーをデコヒーレンスエラーの指標として選んだ．また，相関がほとんどなかった測定以外のゲート操作中のエラーについてもマッピング指標から外す．最終的に，論理エラー率との相関が強かった測定のデコヒーレンスエラー，測定操作中のエラー，iOlius 等の提案手法の 3 種類をマッピング指標とした．

これらのマッピング指標を用いて，ノイズ適応型配置の目的関数を定義する．上のマッピング指標はいずれも小さいほど，論理エラー率が小さくなることがわかっている．そこで，最終的にはこれらの指標の線形結合を最小化する箇所にマッピングを行う．しかし，これまでの評価だけでは目的関数における 3 つのマッピング指標の係数が定まらない．そこで，今回は符号距離，シンドローム測定サイクル，初期状態の条件ごとに重回帰分析を行った．マッピング指標間の関係性を明確化するために，相関係数が最大だった測定のデコヒーレンスエラーに基準を置き，その係数を 1 として，他の指標の係数を相対的な重み係数となるように係数を再計算した．そして，条件ごとに再計算した各マッピング指標の係数について，中央値をとった．その結果から，表面符号のノイズ適応型配置の目的関数を式 7 として定義した．この式において，測定のデコヒーレンスエラーは $f_{e_{m,t}}$，測定操作中のエラーは $f_{e_{m,all}}$，iOlius 等の提案手法は f_{map} とする．

$$F_{cost} = 1 \cdot f_{e_{m,t}} + 0.41 \cdot f_{e_{m,all}} + 0.12 \cdot f_{map} \qquad (7)$$

本稿では式 7 の F_{cost} を最小化する箇所をノイズ適応型配置における最適なマッピングと定義した．

6. ノイズ適応型配置の性能評価

6.1 評価環境

本章では 5.3 節で定義した表面符号のノイズ適応型配置の性能評価を行った．基本的な評価環境は 5.2.1 節の条件に準じる．この性能評価では，4 章で述べた方法により，5 回分の実機のキャリブレーションデータからそれぞれ 5 種類のランダムなノイズモデルを生成し，使用した．また，すべてのマッピングの候補について論理量子ビットを構成し，論理エラー率を比較するのではなく，ランダムに 20 か所のマッピング候補を抽出し，それらとノイズ適応型配置のマッピングと比較した．

物理量子ビット数　400（20×20 の格子）
符号距離　3，5，7
シンドローム測定サイクル　符号距離数（3 または 5 または 7），25
データ量子ビットの初期状態　$|0\rangle$，$|+\rangle$
#shots　1000000

上の条件により，ランダムなマッピングとノイズ適応型配置の性能について，論理エラー率の観点で評価を行った．

6.2 評価結果

符号距離やシンドローム測定サイクル，初期状態の条件ごとのランダム配置またはノイズ適応型配置における論理量子メモリ性能の評価結果を表 4 に示す．表 4 の論理エラー率やランダム配置に対する改善率等の論理エラー率をもとに計算する指標は，複数生成したノイズモデルの平均値である．

ノイズ適応型配置のランダム配置に対する改善率（以下，ノイズ適応型配置による改善率）はノイズ適応型配置とランダム配置の論理エラー率の差をランダム配置の論理エラー率で割ったものである．すなわち，この指標はノイズ適応型配置による，論理エラー率の低減効果の大きさを示す．表 4 のノイズ適応型配置による改善率に着目すると，ノイズ適応型配置はランダム配置に対して最小で 12%，最大で 21%論理エラー率を低減した．また，すべての条件全体での改善率は平均で約 15%であり，符号距離別では，符号距離 5 の時の改善率が高く，次いで符号距離 3 の時の改善率が高くなった．この指標の結果から，ノイズ適応型配置による平均的な論理エラー率の低減効果の大きさが約 15%であることがわかった．

ノイズ適応型配置のエラー率低減可能性は，生成したノイズモデルの全数に対してノイズ適応型配置による性能改善が見られたノイズモデルの割合を表す．性能改善の頻度が高いほど，ノイズモデルによらず性能を改善できることを示し，この指標が 50%の場合はランダム配置と同程度の性能となると予想される．すなわち，この指標はノイズ適応型配置による論理エラー率の低減効果が見られる頻度を示す．表 4 をみると，最小でも 64%，最大では 92%の確率で，ランダム配置よりも論理エラー率が低い．すべての条件全体では平均 79%の確率でランダム配置よりも論理エラー率が低い．また符号距離ごとにその平均を確認すると，符号距離 3 の時には 89%，符号距離 5 の時には 81%，符号距離 7 の時には 67%となった．すなわち，符号距離が小さい時ほど，ノイズ適応型配置による性能改善がみられる可能性が高い．

符号距離が小さいほど性能改善が見られたのは，ノイズ適応型配置の目的関数を定義する際に符号距離 3，5 のみでマッピング指標と論理エラー率の重回帰を行ったことが原因である．

以上の結果からランダム配置と比較した際のノイズ適応型配置による性能改善効果について確認した．しかし，これだけでは本稿で定義したノイズ適応型配置の目的関数がどの程度マッピングを最適化できているのかは不明である．そこで，マッピングによる性能改善の最大値に対して，ノイズ適応型配置による性能改善の割合を評価した．これにより，本稿で定義した目的関数の性能を評価する．

前提として，マッピングによる性能改善はランダム配置における最小の論理エラー率の時に最大になると仮定する．まず，最善の配置のランダム配置に対する改善率を最善の配置とランダム配置の論理エラー率の差を，ランダム配置の論理エラー率で割った値として定義する．これはマッピングの工夫による性能改善の上限を与える．この指標に対するノイズ適応型配置の改善率の割合によって，本稿で定義したノイズ適応型配置の目的関数の性能改善が上限にどれだけ近く，最適化されているかがわかる．

図 3 に各条件におけるノイズ適応型配置の最適化率を示す．最適化率は最大で 66%，最小は 26%となった．そして，符号距離 3 の時は平均約 59%，符号距離 5 の時は平均約 52%，符号距離 7 の時は平均約 30%となった．この結果から符号距離 3 の場合は最低でも 52%程度まで最適化できている．そして，符号距離 5 の場合は最低でも 45%程度まで最適化できている．一方で，符号距離 7 の場合は最低だと 26%しか最適化されていない．すなわち，符号距離 3，5 の場合は目的関数が比較的フィットしており，最適化の上限と比較して，その半分程度の性能改善ができている．それに対して符号距離 7 の場合は目的関数が十分にフィットしていないことから，最適化率が他の 2 つの条件と比較して低い．すなわち，符号距離 7 に対してはノイズ適応型配置の目的関数の改善によるさらなる性能向上の余地が大きい．また，符号距離 3，5 についてもまだノイズ適応型配置の目的関数の改善の余地はある．

本稿では符号距離 3，5 の場合のみでノイズ適応型配置の目的関数を定義したが，さらに符号距離を大きくしたときも含めて目的関数を定義することにより，符号距離 7 の

表 4　ランダム配置またはノイズ適応型配置における論理量子メモリ性能の評価結果

符号距離	初期状態	シンドローム測定サイクル	論理エラー率 [%]			ノイズ適応型配置のランダム配置に対する改善率 [%]	ノイズ適応型配置のエラー率低減可能性 [%]
			ノイズ適応型配置	ランダム配置	差		
3	$\|0\rangle$	3	2.0067	2.3689	0.3621	15.29	92
3	$\|0\rangle$	25	9.3651	10.9308	1.5656	14.32	88
3	$\|+\rangle$	3	2.1186	2.5296	0.4111	16.25	92
3	$\|+\rangle$	25	8.5321	9.9519	1.4198	14.27	84
5	$\|0\rangle$	5	0.2821	0.3348	0.0526	15.72	88
5	$\|0\rangle$	25	0.6881	0.8660	0.1779	20.54	80
5	$\|+\rangle$	5	0.4191	0.4914	0.0723	14.72	76
5	$\|+\rangle$	25	0.8997	1.0900	0.1903	17.46	80
7	$\|0\rangle$	7	0.0539	0.0619	0.0080	12.87	68
7	$\|0\rangle$	25	0.1059	0.1240	0.0181	14.58	68
7	$\|+\rangle$	7	0.0937	0.1085	0.0148	13.64	68
7	$\|+\rangle$	25	0.1605	0.1830	0.0225	12.30	64

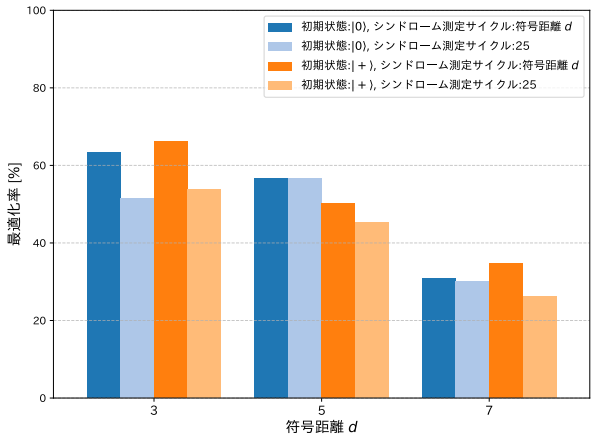

図 3　ノイズ適応型配置の最適化率

場合についてもノイズ適応型配置のエラー率低減可能性や目的関数の最適化率を向上できる．また，重回帰によりノイズ適応型配置の目的関数を定義したが，データ数を増やすことで機械学習等による目的関数の定義も可能になり，さらなる性能向上の可能性がある．

7. 関連研究

7.1 NISQ におけるノイズ適応型配置

現在実現されているゲート方式の量子コンピュータは NISQ と呼ばれるものである．NISQ も FTQC と同様に実機のエラー率やコヒーレンス時間などは物理量子ビットごとに異なる．そこで，計算結果におけるエラーの影響を最小化するための手法の一つとして，エラー率の低い物理量子ビットを使用するノイズ適応型配置が提案されている [14]．文献では CNOT ゲートや測定ゲートなどのエラー率の大きいゲート操作を対象として，信頼性を計算し，そのスコアを最大化するところにマッピングを行う．その結果，接続性だけを最適化する場合よりも試行成功率が向上することが示されている．このような NISQ におけるノイズ適応型配置は，既に量子 SDK である Qiskit のトランスパイラにおいて実用化されており、実機の NISQ における回路実行の際の標準的な手法となっている。

7.2 非一様な物理エラーを前提とする表面符号

2 章にて，従来の研究は物理量子ビットのエラー率が一様であることを前提とするものが多いと述べた．ここでは非一様な物理エラー環境を前提とする先行研究について取り上げる．

7.2.1 配置等の工夫による論理エラー率の低減

iOlius 等は実機のキャリブレーションデータを使用した配置の最適化を提案する [5]．この文献では論理量子ビットエラーを防ぐための手法として，データ量子ビットの物理エラー率の性能が交互になるようにすることを提案している．これによりエラー鎖が長くなることを防ぐことができ，論理エラー率を低減することができる．また，物理エラー率に応じて MWPM のエッジの重みを変動させる手法により，復号性能が向上した．

この文献ではエラーの原因としてデコヒーレンスしか考慮されていない．また，あらかじめエラー率が定まった物理量子ビットの中からマッピングを決めるのではなく，論理エラーを防ぐために，論理量子ビットに対して物理量子ビットのエラー率が高いものと低いものが交互となるように後から物理量子ビットのエラー率を仮定している．すなわち，この文献はマッピングの際のデータ量子ビットの物理エラー率がどのような分布になると，論理エラー率が低減できるかを理論的に示したものである．

7.2.2 パルスレベルでの最適化による論理エラー率の低減

文献 [6] において Google の実機である Sycamore QPU において，閾値定理の閾値を超える性能を実現した．従来の実機では物理量子ビットにおけるエラーが大きかったため，符号距離を大きくしたことによる誤り訂正能力の向上よりも，使用する物理量子ビットのエラーの影響の方が大きくなっていた．そのため，符号距離を大きくした際にむ

しろ論理エラー率が高くなっていた．そこで，Sycamore QPU に Snake Optimizer 最適化を用いることで，符号距離を大きくしたときに，誤り訂正能力の向上を物理量子ビットのエラーの影響よりも大きくすることができた．文献では実機の QPU について物理レベルのデータも含めて，膨大なキャリブレーションデータを使用し，ゲート操作のパルスの形状，振幅，duration time，周波数などのパラメータを最適化する．この手法はパルスレベルでの制御の最適化にあたり，高精度な論理量子ビットの実現が可能である．一方で，一般的なユーザには物理レベルのキャリブレーションデータの取得や動作の最適化は困難である．

8. まとめ

本稿では非一様な物理エラーを前提に，FTQC の表面符号の論理エラー率を低減する手法としてキャリブレーションデータを用いたノイズ適応型配置を提案し，その性能について評価した．その結果，本稿で定義したノイズ適応型配置により，ランダム配置に対する改善率は最大で約 21%，平均で約 15%となり，論理エラー率の低減効果を確認できた．そして，エラー率低減可能性は符号距離 3 の時に最大の 92%となったが，符号距離が 7 の場合は最大でも 68%にとどまった．さらにノイズ適応型配置とマッピングの工夫による性能改善の上限を比較し，最適化率を評価した結果，符号距離 3，5 の場合は最適化率は平均 50%を超えた．しかし符号距離 7 の場合の最適化率は約 30%であった．よって本稿では，表面符号のノイズ適応型配置には論理エラー率の低減効果があることが確認できた．しかし，ノイズ適応型配置の目的関数には改善の余地がある．そのため，モデルの改善等により，ノイズ適応型配置によるさらなる論理エラー率低減の可能性がある．

参考文献

[1] Shor, W. P.: Scheme for reducing decoherence in quantum computer memory, *Phys. Rev. A*, Vol. 52, No. 4, pp. R2493–R2496 (1995).
[2] Shor, W. P.: Fault-tolerant quantum computation, *Proc. 37th Conference on Foundations of Computer Science*, IEEE, pp. 56–65 (1996).
[3] Bravyi, B. S. and Kitaev, Y. A.: Quantum codes on a lattice with boundary, arXiv (1998).
[4] Fowler, G. A., Mariantoni, M., Martinis, M. J., et al.: Surface codes: Towards practical large-scale quantum computation, *Phys. Rev. A*, Vol. 86, No. 3, p. 032324 (2012).
[5] iOlius, A. d., Martinez, J. E., Fuentes, P., et al.: Performance of surface codes in realistic quantum hardware, *Phys. Rev. A*, Vol. 106, No. 6, p. 062428 (2022).
[6] Acharya, R., Aleiner, I., Allen, R., et al.: Suppressing quantum errors by scaling a surface code logical qubit, *Nature*, Vol. 614, No. 7949, pp. 676–681 (2023).
[7] Acharya, R., Abanin, A. D., Aghababaie-Beni, L., et al.: Quantum error correction below the surface code threshold, *Nature*, Vol. 638, No. 8052, pp. 920–926 (2025).
[8] Steane, M. A.: Error Correcting Codes in Quantum Theory, *Phys. Rev. Lett.*, Vol. 77, No. 5, pp. 793–797 (1996).
[9] Aharonov, D. and Ben-Or, M.: Fault-Tolerant Quantum Computation with Constant Error Rate, *SIAM Journal on Computing*, Vol. 38, No. 4, pp. 1207–1282 (2008).
[10] Breuckmann, P. N. and Eberhardt, N. J.: Quantum Low-Density Parity-Check Codes, *PRX Quantum*, Vol. 2, No. 4, p. 040101 (2021).
[11] Wang, S. D., Fowler, G. A. and Hollenberg, C. L. L.: Surface code quantum computing with error rates over 1%, *Phys. Rev. A*, Vol. 83, No. 2, p. 020302 (2011).
[12] Bao, F., Deng, H., Ding, D., et al.: Fluxonium: An Alternative Qubit Platform for High-Fidelity Operations, *Phys. Rev. Lett.*, Vol. 129, No. 1, p. 010502 (2022).
[13] Hyyppä, E., Kundu, S., Chan, C. F., et al.: Unimon qubit, *Nature Communications*, Vol. 13, No. 1, p. 6895 (2022).
[14] Murali, P., Baker, M. J., Javadi-Abhari, A., et al.: Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers, *Proc. the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)*, Association for Computing Machinery, pp. 1015–1029 (2019).