Vol.2022-QS-6 No.17
2022/7/8

表面符号と格子手術を用いた量子計算のためのコンパイラバックエンド開発

脇坂 遼[1,2,a)]　鈴木 泰成[2,3,b)]　徳永 裕己[2,c)]

概要：表面符号と格子手術はそれぞれ量子誤り訂正符号およびそれを用いた計算プロトコルであり、大規模な量子計算に必要となる誤り耐性量子計算を実現するものとして近年注目されている。符号化された論理量子ビットに対して各論理操作を施すためには複雑な低級操作が要求されるため、人間が直接記述することは困難である。したがってその活用にはユーザーが記述したプログラムを実行可能な低級操作列に翻訳するコンパイラが必要となる。また、より良いコンパイルアルゴリズムの探究のためには、コンパイル後のプログラムの性能や安全性を検証するためのソフトウェア基盤も必要である。本研究ではこうした解析を可能とするため、表面符号と格子手術のためのコンパイラバックエンドを開発した。本研究ではコンパイルアルゴリズムを開発しただけでなく、コンパイル後のプログラムを表現するための対象言語を形式的に与え、対象言語を用いたプログラム解析器を実装した。さらに既存のベンチマーク用量子プログラムに対してコンパイルを行い、様々な条件下で性能評価を行った。その結果、コンパイラの最適化の恩恵を最大限に利用するためには、プログラムを記述する段階で命令レベル並列性を高める必要があることが明らかとなった。

1. はじめに

近年、古典計算を超えうる計算パラダイムとして量子計算が注目され、小規模な量子計算機 (Noisy Intermediate-Scale Quantum, NISQ) が実現した。NISQ デバイスでは量子誤りの影響をそのまま受けるため、実用上必要とされる大規模量子プログラムを動作させるには量子誤り訂正符号を用いた誤り耐性量子計算 (Fault-tolerant Quantum Computation, FTQC) が必要である。

表面符号 [1] は、FTQC を実現するための論理量子ビットを構成する量子誤り訂正符号の 1 つとして有力視されている。表面符号で符号化された論理ビットに対する操作でボトルネックとなるのが、T-gate と呼ばれる 1 論理量子ビットの操作である。従って、表面符号を用いて行う量子計算に要する時間は T ゲートを実行する回数である T-count や、並列化した際に何個の T-gate が必要となるかを表す T-depth で表現される。また、表面符号によって構成された論理ビット間で CNOT ゲートや多体測定などの論理演算を行う方法として格子手術 [2] が提案されている。これらを組み合わせることで、複数の表面符号で符号化された論理量子ビットを用いた万能量子計算が可能となっている。

表面符号を用いて行う FTQC の課題は、ユーザーが記述する量子プログラムとのギャップである。格子手術で行われる低級な論理操作は複雑であり、人間が直接記述することは難しい。さらに、プログラムを効率的に実行するためには命令をできるだけ並列に実行することが求められるが、格子手術で行われる命令の並列性をどのようにして向上させるべきかは明らかではない。したがって表面符号の効率的な活用のためには、記述された量子プログラムを格子手術での操作列に翻訳するコンパイラが必須となる。こうした状況を踏まえ、表面符号と格子手術を対象としたコンパイル技法は様々なものが提案されてきている [3], [4], [5] が、コンパイラの評価基盤が成熟しておらず、提案されたアルゴリズムの比較検討を行うことが難しいという問題がある。また表面符号や格子手術を記述するのに適したプログラミング言語が整備されておらず、コンパイル後のプログラムの再利用性や形式的なプログラム解析も困難である。今後実用的な量子アルゴリズムを表面符号を用いて実装および評価するためには、そのためのプログラミング言語を含むコンパイラ基盤を構築する必要がある。

[1] 京都大学大学院情報学研究科
[2] NTT コンピュータ&データサイエンス研究所
[3] JST さきがけ
〒 1332-0012 埼玉県川口市本町 4-1-8
[a)] wakizaka@fos.kuis.kyoto-u.ac.jp
[b)] yasunari.suzuki.gz@hco.ntt.co.jp
[c)] yuuki.tokunaga.bf@hco.ntt.co.jp

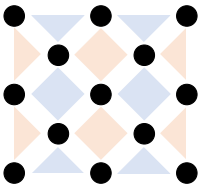

図 1 符号距離 3 の表面符号の構成。黒い丸はデータ量子ビットを、赤と青の面はパウリ X,Z でのスタビライザー測定を表す。

本研究の貢献

本研究では、表面符号と格子手術のためコンパイラバックエンドを開発した。まず、格子手術を表現できる低級な命令セットをもつ対象言語を形式的に設計した後、ユーザーが記述した量子プログラムを対象言語へコンパイルするアルゴリズムを構築した。コンパイルは、プログラムを動かす量子計算機のアーキテクチャ情報を用いて行われる。次に、コンパイル後のプログラムの性能評価を行うため、対象言語を対象としたプログラム解析器を実装した。これにより、アーキテクチャ制約等を無視した指標である T-count や T-depth に比べより実際的な性能評価が可能となった。最後に、実際にいくつかのベンチマーク用の量子プログラムをコンパイルし、表面符号と格子手術を用いて実行した際の実行サイクル数の解析を行った。その結果、コンパイルアルゴリズムの恩恵を受けるためには、元々のプログラムが持っている命令の並列度を高める必要があることを確認した。

本研究で開発したコンパイラバックエンドは、今後様々な量子プログラムを表面符号と格子手術を用いた形式にコンパイルし評価する際のソフトウェア共通基盤として利用することができる。

2. 背景

2.1 表面符号と格子手術

高信頼かつ大規模な量子計算を実現するためには、量子誤り訂正符号を用いた誤り耐性量子計算が必要である。量子誤り訂正符号の中でも、表面符号 [1], [6] は現実のデバイスで実装できる可能性が高く、かつ、性能の良い符号として注目されている。表面符号では、図 1 に表されるように複数の物理量子ビットを二次元格子状に配置して 1 つの論理量子ビットを構成する。量子計算機上では、物理量子ビットが二次元格子状に大きく並んでいる一部分を使って 1 つの論理ビットを構成し、それを複数並べることによって複数の論理量子ビットを同時に扱う。（近似的に）万能な量子計算を実現するためには、万能ゲート集合 $\{H, CX, T\}$ ゲートが実現できれば良いことがよく知られている。以降は表面符号と格子手術によって万能ゲート集合の各操作がどのように実現されるかを簡単に説明する。

H ゲートは transversal な操作であり、符号を構成する物理量子ビット全てに H ゲートを適用することで実現できる。実際には H ゲートの適用によって符号の X と Z の意味が反転するため、それを元に戻すための論理ビットの拡張や移動といった細かい操作を伴うが、ここでは省略する。

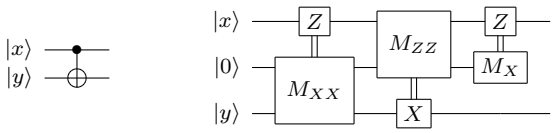

図 2 CX ゲートの分解を行う量子回路

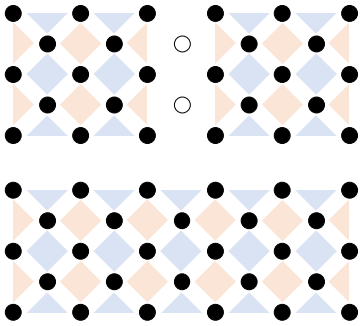

図 3 2 体測定 M_{XX} の実現。図では、二つの符号距離 3 の表面符号を、符号に利用されていない量子ビット (白い丸) を一時的に利用して下段のように merge し、再度上段のように split することで論理測定を実施することができる。

CX ゲートは図 2 のように 3 回の測定操作を含む操作列に分解できる [7]。ここで登場する 2 体測定 M_{ZZ} および M_{XX} は格子手術 [7] によって実現できる。例えば M_{XX} は図 3 のように実現することができる。図 3 では 2 つの論理ビットが近い位置に隣接しているが、離れた位置にある場合でも $|0\rangle$ 初期化される部分を伸ばして繋げることで同じ操作が実現可能である。

T ゲートは魔法状態と呼ばれる量子状態 $|A_L\rangle$ を消費するゲートテレポーテーションによって実現される。T ゲートを高い精度で適用するためには魔法状態蒸留 [8], [9] など負荷の大きい操作を通して高忠実度な魔法状態 $|A_L\rangle$ が必要となるため、T ゲートは他の命令に比べレイテンシが大きく、量子プログラムの実行時間の大半を占める。量子プログラムの最適化において、プログラムが利用する T ゲートの数である T-count や、回路における T ゲートの深さを表す T-depth といった指標が用いられるのはこうした事情に基づいている。なお、T ゲートのゲートテレポーテーションでは S ゲートの適用が必要であるため、コンパイラの命令セットは S ゲートを適用できる表現力が求められることには注意が必要である。S ゲートの実現は Y 測定を測定型量子計算でテレポートする方法や [10]、S ゲート用の魔法状態を導入するといった方法 [7] がある。

2.2 命令の並列実行可能性

複数の多体操作を並列に実行する場合は、マージ操作で使用する量子ビットが互いに干渉しないようにしなければならない。したがって量子プログラムの最適化にあたっては、単に T-depth や T-count を小さくするだけでなく、命令の並列度を高めるようにマージのパスを上手く選択する必要がある。また、魔法状態の供給量も命令の並列度に大きく影響する。例えば魔法状態が同時に高々 M 個しか供給できないならば、T ゲートの並列度を M より大きくすることは不可能である。量子プログラムの最適化で重要視される T-depth は T ゲートの並列度を無限大として計算されるものであり、この点で実際の量子計算機上で動作させる際の性能とは多少なりともギャップが存在する。

3. 関連研究

表面符号を用いた量子プログラムのリソース解析という観点では、QuRE [11] や OpenSurgery [12] がある。QuRE では様々なアーキテクチャに対応している一方で、実行時間の解析では各操作の並列実行可能性などが考慮しきれていない。OpenSurgery では命令の並列実行を含めた高度な最適化とリソース解析を行っているが、コンパイル後のプログラムが独自の 3D ブロック表現を用いており、ユーザーが解析を行うことが難しい。本研究では低級言語に基づく体系的かつ正確な性能の解析を行い、プログラムの持つ性質がコンパイル時の最適化に与える影響をも調査している。

格子手術とは別の計算方法である defect braiding を対象としたコンパイラおよびシミュレーターとして ScaffCC とそれに付随する braidflash がある [13]。本研究は ScaffCC と braidflash と類似ツールを格子手術向けに実装したものと捉えることもできるが、対象言語を形式的に定義し、対象言語に基づく解析を行おうとする点は異なっている。

表面符号および格子手術のコンパイル時最適化の研究には Beverland らによるものがある [5]。Beverland らは格子手術のパススケジューリングのために最大流や量子テレポーテーションを用いたアルゴリズムを与えており、本研究にも一部組み込まれている。

近年、プログラミング言語的なアプローチにより、コンパイル後のプログラムに対する検証やコンパイルアルゴリズム自体の正しさを検証する試みが盛んに行われている [14][15][16][17]。同様のアプローチは表面符号や格子手術でのコンパイルにおいても有効である。本研究では表面符号と格子手術のための低級言語を与えているため、その上に型システムなどの静的解析手法を実装することによってコンパイル後のプログラムやコンパイル自体の信頼性を高めることが期待できる。

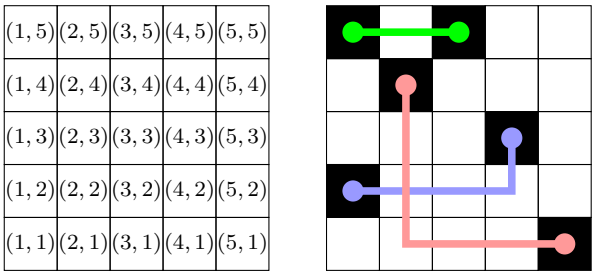

図 4 論理セルのレイアウト（左図）およびマージ操作のパス（右図）。右図において $(2,4)$-$(5,1)$ パスと $(1,2)$-$(4,3)$ パスは互いに干渉するため、並列実行できない。

表 1 命令毎のレイテンシ（d は符号距離）

Init	Pauli	H	Meas	S	T
1	d	$3d$	d	$14d$	$17d$

4. コンパイラバックエンド

本研究で実装したコンパイラバックエンドは、表面符号と格子手術のための命令セットを含む低級プログラミング言語および OpenQASM 等の中間表現からのコンパイルアルゴリズムで構成される。また、コンパイル後の低級なプログラムに基づく実行サイクル数を推定する評価器も備えており、低級言語を対象とする任意のコンパイルアルゴリズムを同一条件下で比較検討できる。以降は、本研究で扱う問題設定について整理した後、コンパイラバックエンドの詳細を説明する。

4.1 問題設定

一般には二次元格子状に並んだ物理量子ビットのどの位置に論理量子ビットを配置するかは自由である。しかしながら、配置の自由度を高めるほどそれを制御するための命令セットや実際の処理が複雑になり、ソフトウェア的観点からは扱いにくいものになってしまう。そこで本研究では簡単のため図 4 のようなグリッドグラフを考え、論理量子ビットはグリッドの 1 つのセルに収まるように配置されると仮定する。また 2 つの論理ビットのマージ操作（二体測定）は、グリッドグラフ上で対象のセルを結ぶパスで表現する。複数のマージ操作が並列実行可能であるとは、それらのパスが互いに disjoint であるときに限る。この仮定は文献 [5] や [10] など表面符号のためのコンパイルアルゴリズムを設計する際に広く用いられている考え方であり、コンパイル時に解く必要のある問題や命令セットを単純にすることができる。

4.2 対象言語

コンパイラが出力する対象言語の文法は以下で定義される。

$$
\begin{aligned}
e &::= \texttt{Wait}\ n \mid x = \texttt{init}(i,j) \mid \texttt{discard}\ x \mid \\
&\quad \texttt{Meas}\ \overline{x,k}\ p \mid U\ x \mid e_1; e_2 \mid e_1 \mid\mid e_2 \\
k &::= X \mid Z \\
p &::= \epsilon \mid (i,j), p \\
U &::= X \mid Z \mid H \mid S
\end{aligned}
$$

式 `Wait` n は n サイクルの間待機する命令であり、後続命令が依存している命令の完了を待機するときに用いる。式 $x = \texttt{init}(i,j)$ は (i,j) の位置に論理量子ビットを $|0_L\rangle$ 状態で生成する。一方で `discard` x は量子ビット変数 x を破棄して再利用可能にする命令である。この命令は明らかに破棄可能であるとわかっている部分にコンパイラが注釈として挿入するためのものである。測定の命令 `Meas` $\overline{x,k}\ p$ では量子ビットの集合 $\bar{x}$ をパス p に沿って多体測定を行う。ここで $\overline{k}$ はそれぞれの量子ビットの測定の種類 (X,Z) を表す。式 $e_1 \mid\mid e_2$ は 2 つの式 e_1, e_2 を並列に実行する。

対象言語において各操作を実行するのに必要なサイクル数（レイテンシ）を表 1 に示しておく。レイテンシは後述するようにコンパイル時のスケジューリングや実行性能評価に用いる。

4.3 コンパイルアルゴリズム

今回実装したコンパイルアルゴリズムを Algorithm 1 に示す。まず初めに、今の状況において即座に発行できる命令を収集する。その中から魔法状態を消費する命令を抜き出し、文献 [5] と同様に最大流アルゴリズムによってパス探索を行う (Algorithm 2)。その後魔法状態を消費しない命令について、現在使用できるセルだけを用いてパス探索を行う。パス探索は命令を 1 つずつ順番に見ていき、幅優先探索により対象のセルを最短距離を結ぶようなパスを選び続けるという戦略を取っている。パス探索の結果パスを見つけられず実行できなかった命令は次回のイテレーションに延期する。

魔法状態に関するパス探索に最大流アルゴリズムを用いている理由は、魔法状態を消費する命令とペア付けする魔法状態の位置は自由に決めて良いからである。また、魔法状態を消費する命令を優先的に発行しているのは、その他の命令に比べてレイテンシが大きいからである。

Algorithm 1 ではレイテンシを考慮してコンパイルを行っていることに注意されたい。すなわち、セル集合 V を使用するレイテンシ l の命令を発行すると、以降 l サイクルの間は V を使用することはできないようになっている。

Algorithm 1 コンパイルアルゴリズム（概要）

入力 C: 入力プログラム、G: セルレイアウトを表すグラフ
入力 V_m: 魔法状態が生成されるノード $\subseteq V(G)$

function COMPILE(C, G, V_m)
  QUBITALLOC(C, G) ▷ 量子ビット変数をセルに割当
  $V \leftarrow \emptyset$ ▷ 使用されているセル集合
  $\overline{e} \leftarrow \epsilon$ ▷ コンパイル後の命令列
  while $C \neq \emptyset$ **do**
    Remove released cells from V ▷ 使用済セルを解放
    $I \leftarrow$ available instructions in C
    $I_s \leftarrow \{e \in I \mid e$ is a single qubit instruction$\}$
    $I_m \leftarrow \{e \in I \mid e$ consumes a magic state$\}$
    $I_c \leftarrow I \setminus (I_m \cup I_s)$
    $I_{\text{issue}} \leftarrow I_s$
    $P \leftarrow$ SEARCHMAGICPATH($V(I_m), V_m, G \setminus V$)
    $V \leftarrow V \cup P$
    Append $\{e \in I_m \mid \exists p \in P.\ e$ corresponds to $p\}$ to I_{issue}
    for $e \in I_c$ **do**
      $\{u, v\} \leftarrow$ the operands of e
      $p \leftarrow$ SEARCHPATH($G \setminus V, u, v$) ▷ 幅優先探索
      if $p \neq$ `None` **then** ▷ パスが存在
        $V \leftarrow V \cup p$
        Push e to I_{issue}
    Remove I_{issue} from C
    Vectorize and push I_{issue} to $\overline{e}$
  return $\overline{e}$

Algorithm 2 魔法状態とのマージパス探索

入力 V: 魔法状態を消費するセル集合
入力 V_m: 魔法状態が格納されているセル集合

function SEARCHMAGICPATH(V, V_m, G)
  $s, t \leftarrow$ virtual source and sink nodes
  $G' \leftarrow G \cup \{(s, s') \mid s' \in V\} \cup \{(t', t) \mid t' \in V_m\}$
  $f \leftarrow$ MAXFLOW(G', s, t) ▷ f は残余グラフ
  return $\{p \setminus \{s, t\} \mid p \in (s$-$t$ paths in $f)\}$

5. 評価

5.1 対象とするアーキテクチャおよびデータセット

本研究では図 5 で表されるアーキテクチャおよび文献 [18] および [19] のデータセットを対象とし、第 4 節で与えたアルゴリズムの性能評価を行った。図 5 において、魔法状態のセルとは別に魔法状態蒸留を行うためのファクトリと呼ばれる領域が本来必要であるが、本研究では簡単のため省略している。加えて、本来であれば S ゲートの実装はいくつかのセルを専有して行われるものであるが、現在の実装ではそれを無視してコンパイルするようになっており、この点は少し不正確である。

なお、アーキテクチャのセルレイアウトは自由に設定できるようになっているため、実際には図 5 以外の異なるアーキテクチャを対象としたコンパイルも可能である。

5.2 評価結果

実験結果を表 2 に示す。今回用いたデータセットの中では、魔法状態の供給量は `gf2^32mult` を除きプログラムの実行サイクル数にほとんど影響を与えなかった。`gf2^32mult` だけが魔法状態の数の影響を大きく受ける理由として、元々のプログラムの命令の並列度が高いことが考えられ

る。gf2^32mult の T-count と T-depth の比率を見ると、T-count に対して T-depth が小さく、T ゲートの命令レベル並列性が高いことを示唆している。

実際の命令レベルの並列度を調査するため、gf2^32mult と ham15 を代表して取り上げ、実行中の魔法状態の消費量および、魔法状態の枯渇やパスの干渉によってストールしている命令数の時間変化を調査した。その結果を図 6 に示す。 ここで、(a) は gf2^32mult を、(b) は ham15 の評価結果を表す。また、(a-1),(b-1) のグラフは魔法状態が無限に存在し、パスの干渉を無視した場合のグラフである。一方、(2), (3) は魔法状態が 5 つのみ供給される条件で実験を行った際に、魔法状態の利用状況と、待機中の命令数をプロットした結果である。この節では得られた表 (a-1), (b-1) の結果と比較することで、表 (a-2), (a-3), (b-2), (b-3) から algorithm 1 が魔法状態をどの程度効率的に消費できているかを考察する。

表 (a-1) と (b-1) の結果から、gf2^32mult は理想的な環境であれば同時に魔法状態を 3 個以上消費する状況も珍しくないが、ham15 はパスの干渉を無視したとしても並列度がほぼ 2 以下になっていることが分かる。したがって ham15 は本質的に命令間の依存が強く、多くの命令は並列に実行不可能である。また表 (a-2) に比べ (b-2) は最大 5 つ存在する魔法状態を効率的に消費できていないが、これは (b-1) で見たように ham15 のプログラム自体の並列度が低いことが理由である。同様の傾向は各サイクルにおいて待機中の命令数を表す (a-3), (b-3) にも見ることができる。表 (a-3) は gf2^32mult は待機中の命令が多く、パス探索の工夫や魔法状態の数を増やすことが実行効率に大きく影響しうることを表している。一方で (b-3) によれば ham15 は待機中の命令自体は常に 2 程度で変化せず、gf2^32mult に比べるとコンパイラの最適化の余地があまり無いプログラムであると考えられる。

なお、ham15 において殆どのサイクルで待機中の命令が存在しているにも関わらずトータルの実行サイクル数が理想的な値と離れていない理由は、待機中の命令が魔法状態を消費しないマージ操作であるために、裏で優先的に動作している T ゲートの適用のレイテンシにうまく隠蔽されているからである。

5.3 議論

今回の評価結果より、コンパイラの最適化の恩恵を受けられるかどうかは元々のプログラムの並列性に大きく影響するということがわかる。従って、例えば古典の加算器で回路段数を小さくするような試みが行われているように、量子プログラムの設計段階から並列性を意識した実装を行う必要がある。記述しているアルゴリズムが消費する量子ビット数が少ない場合は必然的に命令間の依存性が高くなるため、量子ビット数に余裕があればあえて補助ビットを

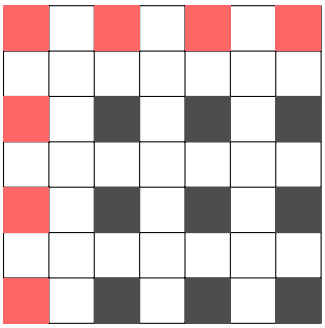

図 5 対象とするアーキテクチャのセルレイアウト。赤いセルには魔法状態が格納され、黒いセルにはデータ量子ビットが配置される。

多く消費することで命令の並列性を向上させることも視野にいれるべきであろう。また、コンパイラはプログラムに隠された並列性を見つけ出し、与えられた計算資源に収まる範囲で並列度を自動的に向上させるといった工夫も必要であると考えられる。

6. 将来の課題

コンパイルアルゴリズム自体にかかるコストの削減は、将来の課題の 1 つである。現在の実装では命令を発行するたびに最大流アルゴリズムや複雑なパス探索を行っているため、大きな量子プログラムに対するコンパイル時間が大きくなる。コンパイルを早くするためには、マージするセルのペアについてあらかじめパスを固定するなど、パス探索のコストを下げなければならない。また、事前にパスを固定するような手法が実行効率に与えるオーバーヘッドも明らかにする必要がある。

別の課題として、論理量子ビットセルの移動、セルの拡大・縮小や魔法状態蒸留のファクトリ制御命令などといった命令セットの拡張がある。拡張によってより正確な性能評価が可能になり、プログラム最適化の幅も広がるが、コンパイルアルゴリズムの複雑化によってコンパイル時間が増大する可能性もある。

また今後行いたいと思っている課題に、コンパイルの正しさの検証手法の確立がある。コンパイルが正しいとは、例えばコンパイル後のプログラムがフォールトトレラントであることと、コンパイル前後でプログラムの意味が保存されることなどのことを指す。本研究ではコンパイル先の対象言語を形式的に定義しているので、文献 [16] のようにプログラムの静的解析手法を導入してコンパイル後のプログラムの信頼性を高められると考えている。

7. 結論

本研究では、表面符号と格子手術のためのコンパイラバックエンドを開発した。コンパイル出力先として十分な命令セットを持つ対象言語を設計し、FTQC を考慮していない入力プログラムを対象言語に変換・最適化するプログ

表 2 コンパイル結果。size はグリッドの縦と横のサイズを、#CC(n) は魔法状態の数が n である条件下でのコードサイクル数を表す。グリッドサイズは必要な量子ビット数を備える最小の正方形のサイズで決定した。

circuit name	qubits	size	T-count	T-depth	#CC(∞)	#CC(5)	#CC(3)
5bitadder	11	9×9	63	36	1261	1261	1261
gf2^32mult_1179_5275	96	21×21	7168	465	43471	80208	86678
ham15_107	16	11×11	3815	1593	78328	78687	79257
rd84_253	16	11×11	5957	2418	121218	121707	122626
life_238	16	11×11	9800	4170	202666	202847	202805
max46_240	16	11×11	11844	4749	240510	241518	242675

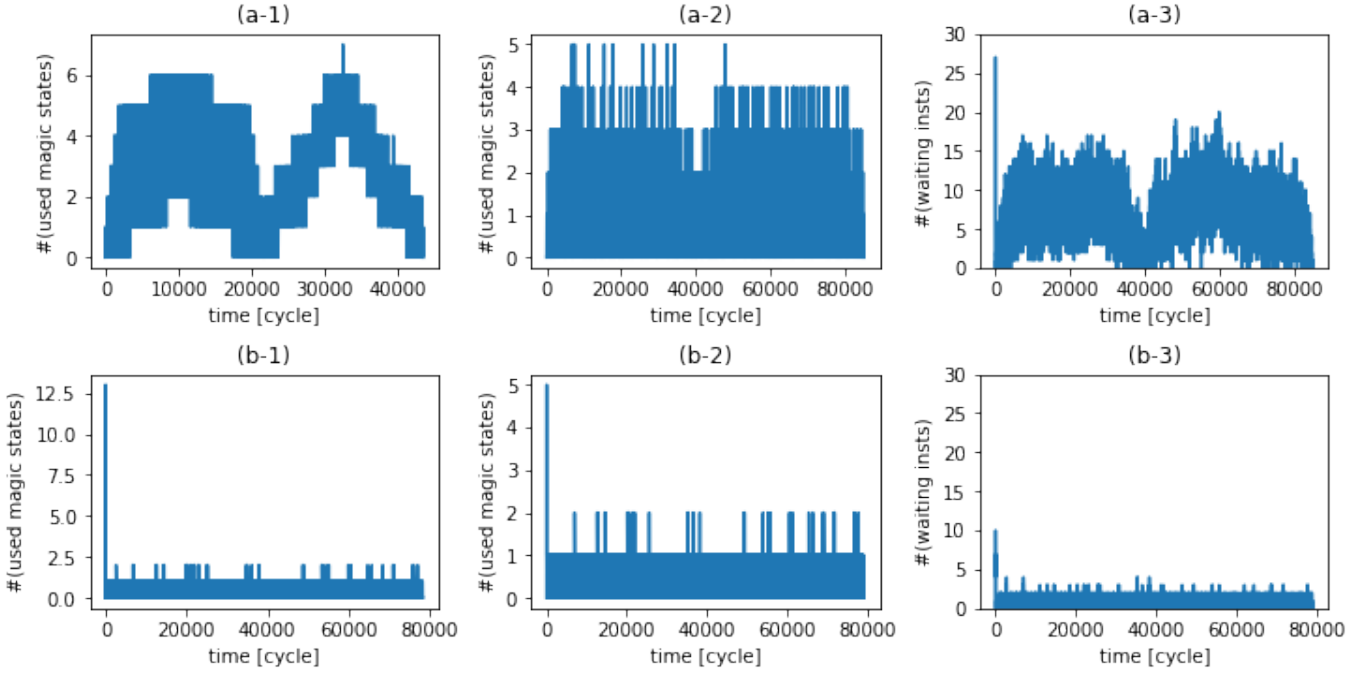

図 6 実行中の魔法状態の利用状況および待機中の命令数の変化。(a) は `gf2^32mult` を、(b) は `ham15` を対象としたもの。(a), (b) ともに 1 つ目のグラフは魔法状態が無限に存在し、パスの干渉を無視した場合のグラフである。また、(2), (3) は魔法状態が 5 つ供給される条件で実験を行った。

ラムを実装した。これによって、入力プログラムを表面符号と格子手術を用いて FTQC を行う際の実行性能評価が可能となった。本研究では実装したアルゴリズムに対して既存のベンチマーク用プログラムを与え、その性能を評価した。

謝辞

本研究は JST さきがけ（助成番号：No. JPMJPR1916），内閣府ムーンショット（助成番号：No. JPMJMS2061）の助成の元で行いました．

謝辞 本研究は JST さきがけ（助成番号：No.JPMJPR1916）、内閣府ムーンショット（助成番号：No.JPMJMS2061）の助成の元で行いました。

参考文献

[1] Bravyi, S. B. and Kitaev, A. Y.: Quantum Codes on a Lattice with Boundary, *arXiv:quant-ph/9811052* (1998).

[2] Horsman, C., Fowler, A. G., Devitt, S. and Meter, R. V.: Surface Code Quantum Computing by Lattice Surgery, *New Journal of Physics*, Vol. 14, No. 12, p. 123011 (online), DOI: 10.1088/1367-2630/14/12/123011 (2012).

[3] Lao, L., van Wee, B., Ashraf, I., van Someren, J., Khammassi, N., Bertels, K. and Almudever, C. G.: Mapping of Lattice Surgery-based Quantum Circuits on Surface Code Architectures, *Quantum Science and Technology*, Vol. 4, No. 1, p. 015005 (online), DOI: 10.1088/2058-9565/aadd1a (2018).

[4] Babbush, R., Gidney, C., Berry, D. W., Wiebe, N., McClean, J., Paler, A., Fowler, A. and Neven, H.: Encoding Electronic Spectra in Quantum Circuits with Linear T Complexity, *Physical Review X*, Vol. 8, No. 4, p. 041015 (online), DOI: 10.1103/PhysRevX.8.041015 (2018).

[5] Beverland, M., Kliuchnikov, V. and Schoute, E.: Surface Code Compilation via Edge-Disjoint Paths, *arXiv:2110.11493 [quant-ph]* (2021).

[6] Fowler, A. G., Mariantoni, M., Martinis, J. M. and Cleland, A. N.: Surface Codes: Towards Practical Large-Scale Quantum Computation, *Physical Review A*, Vol. 86, No. 3, p. 032324 (online), DOI: 10.1103/PhysRevA.86.032324 (2012).

[7] Fowler, A. G. and Gidney, C.: Low Overhead Quantum Computation Using Lattice Surgery, *arXiv:1808.06709 [quant-ph]* (2019).

[8] Knill, E.: Fault-Tolerant Postselected Quantum Computation: Schemes, *arXiv:quant-ph/0402171* (2004).

[9] Bravyi, S. and Kitaev, A.: Universal Quantum Computation with Ideal Clifford Gates and Noisy Ancillas, *Physical Review A*, Vol. 71, No. 2, p. 022316 (online), DOI: 10.1103/PhysRevA.71.022316 (2005).

[10] Litinski, D.: A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery, *Quantum*, Vol. 3, p. 128 (online), DOI: 10.22331/q-2019-03-05-128 (2019).

[11] Suchara, M., Kubiatowicz, J., Faruque, A., Chong, F. T., Lai, C.-Y. and Paz, G.: QuRE: The Quantum Resource Estimator Toolbox, *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pp. 419–426 (online), DOI: 10.1109/ICCD.2013.6657074 (2013).

[12] Paler, A. and Fowler, A. G.: OpenSurgery for Topological Assemblies, *arXiv:1906.07994 [quant-ph]* (2020).

[13] Javadi-Abhari, A., Gokhale, P., Holmes, A., Franklin, D., Brown, K. R., Martonosi, M. and Chong, F. T.: Optimized Surface Code Communication in Superconducting Quantum Computers, *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 692–705 (online), DOI: 10.1145/3123939.3123949 (2017).

[14] Paykin, J., Rand, R. and Zdancewic, S.: QWIRE: A Core Language for Quantum Circuits, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, New York, NY, USA, Association for Computing Machinery, pp. 846–858 (online), DOI: 10.1145/3009837.3009894 (2017).

[15] Hietala, K., Rand, R., Hung, S.-H., Wu, X. and Hicks, M.: Verified Optimization in a Quantum Intermediate Representation, *arXiv:1904.06319 [quant-ph]* (2019).

[16] Hietala, K., Rand, R., Hung, S.-H., Wu, X. and Hicks, M.: A Verified Optimizer for Quantum Circuits, *Proceedings of the ACM on Programming Languages*, Vol. 5, No. POPL, pp. 37:1–37:29 (online), DOI: 10.1145/3434318 (2021).

[17] Li, L., Voichick, F., Hietala, K., Peng, Y., Wu, X. and Hicks, M.: Verified Compilation of Quantum Oracles, *arXiv:2112.06700 [quant-ph]* (2022).

[18] Maslov, D.: Reversible Logic Synthesis Benchmarks Page, https://reversiblebenchmarks.github.io.

[19] Zulehner, A., Paler, A. and Wille, R.: Efficient Mapping of Quantum Circuits to the IBM QX Architectures, *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1135–1138 (online), DOI: 10.23919/DATE.2018.8342181 (2018).