卒業論文　　2013年度（平成25年度）

# ユニバーサル秘密量子計算におけるコンパイラの検証

慶應義塾大学 環境情報学部

氏名：福山　翔一朗

担当教員

慶應義塾大学 環境情報学部

村井 純

徳田 英幸

楠本 博之

中村 修

高汐 一紀

Rodney D. Van Meter III

植原 啓介

三次 仁

中澤 仁

武田 圭史

平成26年1月20日

卒業論文要旨 - 2013 年度 (平成 25 年度)

# ユニバーサル秘密量子計算におけるコンパイラの検証

本研究は、秘密量子計算において利用される Brickwork state アーキテクチャに使用される量子ゲート、量子ビット数の最適化検証を行った。

秘密計算 (Blind Computation) は、既存のクライアント・サーバ方式に加えて、サーバ上での処理を含めた全ての内容を暗号化した状態で演算を行う。そのサーバに量子コンピュータを活用したシステムが、秘密量子計算 (Blind Quantum Computation) である。秘密計算によって、サーバ内に盗聴者が存在してもその内容を傍受されることがない。秘密量子計算は、主に測定ベース量子計算 (Measurement Based Quantum Computation) の一種である Brickwork state アーキテクチャを活用して演算を行う。

本研究では、秘密量子計算において利用される Brickwork state アーキテクチャの演算に利用する量子ビットの数を最適化するための検証として、既存の MBQC アーキテクチャと比較を行う。検証にあたって、AQUA プログラム上で動作する本研究の目的言語である MBQC アーキテクチャ、Brickwork state アーキテクチャに対応できるよう拡張する。この拡張した AQUA プログラムによって、AQUA 言語で記述された大規模な量子回路を Brickwork state アーキテクチャに変換し、大規模な Brickwork state アーキテクチャに役立てることができる。

本研究の検証によって、SWAP ゲートを用いた非隣接 CNOT ゲートにおいて既存の MBQC アーキテクチャよりも効率よく量子ビットを扱える手法を発見した。この最適化手法は AQUA 言語に組み込んだため、量子回路の Brickwork state への AQUA プログラムを用いたコンパイル時に利用できる。また、AQUA 言語の拡張によって動作する MBQC アーキテクチャ、および Brickwork state アーキテクチャの内容を定性的に確認した。

本研究は、今後の大規模なユニバーサル秘密量子計算を開発するにあたって、より効率の良い量子回路を扱う場合に重要である。

キーワード
1. 量子コンピュータ, 2. 秘密量子計算, 3. 量子ネットワーク, 4. 測定ベース量子計算, 5. コンパイラ

慶應義塾大学 環境情報学部

福山　翔一朗

Abstract of Bachelor's Thesis - Academic Year 2013

# Verification of the Compiler in Universal Blind Quantum Computation

This thesis presents the optimization of the number of quantum bits in the Brickwork state used in Blind Quantum Computation(BQC).
Blind Computation performs a calculation a server while keeping all data and even the function encrypted. The important thing is that "The server learns nothing". In addition, Blind Quantum Computation use qunatum computer on Blind Computation. The Brickwork state architecture, is a type of Measurement Based Quantum Computation(MBQC)is suitable for Blind Quantum Computation.
I proposed new optimizations for the number of quantum bits Brickwork state architecture. After verifying these optimizations, I extended the AQUA language to add MBQC adn the brickwork state as target languages. MBQC architecture and Brickwork state architecture which can run on AQUA program. By this extension, programs written in the Exisiting AQUA languange can be converted into Brickwork state quantum circuit architecture.
I have found a method to handle the quantum bits more efficiently than MBQC architecture existing in non-adjacent CNOT gate using the SWAP gate. This optimization can be used in the AQUA language, access when we compile from exisiting AQUA language to Brickwork state architecture. In addition, I have qualitatively confirmed the contents of the Brickwork state architecture and MBQC architecture, operated by the expansion of AQUA language.

Keywords :
1. quantum computer, 2.blind quantum computation, 3.quantum network,
4.measurement based quantum computation, 5. compiler

Keio University, Department of Environment and Information Studies
Shoichiro Fukuyama

# 目 次

# 図 目 次

# 表目次

# 第1章　序論

## 1.1　背景

クラウドサービスなどを代表とするように、私たち（クライアント）はインターネット上で高性能な計算マシン（サーバ）を利用している。これは、クライアント・サーバモデルと呼ばれており、人々は特定の役割を集中的に処理することが可能なサーバを活用することで、端末の本来の計算能力以上の演算を行える。クライアントは、サーバに要求を投げ、サーバはそれに応答する形で通信が行われる。

しかし、要求から応答までの一連の流れの中で、通信路にいつ、どこで盗聴者が現れるかはわからない。そのため、我々は通信内容を秘匿化してやり取りできるサービスを理想と考え、暗号化通信を行う。暗号化通信は、現代の機密情報・個人情報を扱うインターネットにおいて重要な役割を占めており、現在の様々な通信技術に適用されている。

一般的な暗号化通信では、データの送受信を行う通信路だけを暗号化されており、サーバ上での処理は復号された状態、つまり生のデータで処理されている。もしサーバ内に盗聴者が存在して、暗号鍵とデータを盗聴された場合は、入力内容/演算内容が漏洩してしまう危険性がある。

秘密計算 (Blind Computation)[1] は、サーバ上でも暗号化したまま演算を行い、入力から通信・演算・出力まで全てを暗号化した状態で計算を行える方式である。サーバ内から見ると、元の入力データ・演算内容は秘匿化され、処理されるので、通信路だけでなく、一連の処理を全て暗号化した状態でクライアントがサービスを利用できることが強みである。これは、暗号鍵を通信路に流す必要がないという点で優れている。秘密計算の実現によって、通信路のどこに盗聴者がいても内容を傍受されないことが期待できる。Gentry の暗号 [1] がその代表と言える。

Gentry の暗号のような方式を用いて、秘密計算を既存のノイマン型コンピュータ上で実現しようとすると、1 つのプログラムを実行するだけでも必要な計算が膨大になる。通常のクライアントサーバで行われる通信の暗号化によるオーバーヘッドに加えて、演算の部分も含めたやり取りの全て暗号化して行うため、通常のクライアント・サーバ計算よりも膨大なオーバーヘッドが上乗せされてしまうことが原因である。この膨大なオーバーヘッドを減らす方法として期待されているのが秘密量子計算 (Blind Quantum Computation) である。

秘密量子計算では、サーバの入力・演算の部分に量子コンピューティングを用いる。量子コンピューティングは、量子力学を利用して現在のコンピューティングとは異なるアルゴリズム・アーキテクチャで計算を行う技術である。活用すると、既存のノイマン型アーキテクチャのコンピューティングと比べて行う計算の量が比較的少なく済むと期待されて

いる。ただし、秘密量子計算を行う場合であっても、秘密量子計算の計算アーキテクチャのベースとなっている測定ベース量子計算において必要な量子ビットの数が数千・数万ビットである。

また、量子コンピュータの開発によって、既存の暗号化通信の代表とされるDiffie-Hellman鍵共有 [2] や、RSA 暗号 [3] は多項式時間で解読可能になる。それらの鍵共有方式は、因数分解を多項式時間で行う計算方法が現段階では存在しないことに依るものであるため、現段階では解読できなくても、量子コンピューティングの実現によって脆弱となる。しかし、秘密量子計算は元の入力内容・演算内容を通信路に一切送信しないため、どのような方法で通信路/サーバの盗聴を行っても復号して元の内容を一意に特定することができないメリットを持っている。

現在、秘密量子計算は 4 量子ビットの間での物理的な実験 [4] に成功している。 また、量子コンピューティング自体も大規模な演算がいよいよ現実的になってきた。2013 年、東京大学によって 1 万光量子ビットの量子テレポーテーション [5] に成功した。今後も量子コンピューティングの規模は更に大きくなっていくことを考えると、大規模な量子コンピューティングを用いた演算を行っていく場合に秘密量子計算を用いることで、内容を盗聴者にどこからも知らずにリソースを利用することができるようになる。

## 1.2 本研究の目的

本研究の目的は、秘密量子計算のための効率的な量子回路の生成、すなわち量子コンピューティングを効率的に行うためのコンパイラの構築である。現在の測定ベース量子計算のシミュレートのためのコンパイラ、およびトランスレータは整備されていない。また、秘密量子計算をシミュレートするための環境も整備されていない。その中で、将来に大規模な量子コンピューティング、秘密量子計算を行っていく場合に、量子ビットの数の制御、すなわち最適化は重要な役割を果たす。この研究は効率的な大規模な量子回路を生成するのに必須であり、将来の大規模量子コンピュータ構築に役立つ。

ここで、大規模な量子コンピュータの量子回路を生成するために、量子プログラミング言語の一種である AQUA 言語 [6] を活用する。AQUA 言語、およびそれを動作させることができる aqua-tools(以降、AQUA プログラム) は、量子コンピューティングのアーキテクチャの開発を支援するために作られたツールである。AQUA プログラムは、入力された AQUA 言語に対応する形式のコードから別形式のコードを生成する。本研究では、既存の AQUA 言語形式で記述されたプログラムを測定ベース量子計算アーキテクチャ、および Brickwork state アーキテクチャに変換するプログラムの実装を行う。これらのコード生成プログラムは、煩雑な量子コンピュータの回路生成を効率よく行うことを可能にする。一連の AQUA プログラムの拡張の中で、最適化に関する検証を行うことで、大規模な量子コンピュータ構築に役立てる。

Brickwork state は、１つの量子回路を１ブロックと見なして実行を行う性質があるが、１つの汎用でありながら未実装の量子ゲートが数多く存在する。SWAP ゲートと呼ばれる

量子ゲートもまたその１つである。汎用ゲートを実装することは、量子回路の生成を手早く効率よく行うことに繋がる。本研究は、SWAP ゲートに着目し、実装・SWAP ゲートを元にした検証を行っていく。

## 1.3 本研究の成果

本研究によって、測定ベース量子計算アーキテクチャ、Brickwork state の量子回路を生成するコンパイラを AQUA プログラムの拡張として実装することができた。これは、秘密量子計算の古典コンピューティング上でのシミュレーションを行う上で重要である。そして、実装したコンパイラの中で、いくつかの最適化モデルを作り、量子ビットの数がどのようにして変化するのか検証をした。その結果、SWAP ゲートを用いた非隣接 CNOT ゲートにおいて Brickwork state が MBQC アーキテクチャよりも少ない量子ビット数で実行することができる結果を得た。これにより、既存のモデルよりも Brickwork state は非隣接 CNOT ゲートを用いてより少ない量子ビットの数で量子ゲートを実行できる。これは、効率よく量子回路を生成することができる一歩である。

## 1.4 本論文の構成

本論文は 6 章から構成される。第 2 章では、本研究の背景となる要素技術を整理する。第 3 章では、第 2 章で述べた要素技術を元に、秘密量子計算のプログラミング上での処理方法について述べる。そして、本研究が取り組む内容を具体化し、本研究の独自性・有用性を述べる。第 4 章では、第 3 章で述べた内容を元にして、AQUA 言語を活用して aqua-tools の拡張をはじめとする実装を行う。第 5 章では、実装した内容を定性的・定量的な側面から評価する。最後に、第 6 章で本論文の結論と，今後の方針を述べる。

# 第2章　要素技術

本章では、本研究に関する要素技術について説明する。

## 2.1　量子情報の基本

量子情報 [7] では、既存の 0 と 1 から成る古典ビットとは異なり、量子状態を用いて表現する量子ビットを使って情報処理を行う。以下に、量子情報の基本的な概念・処理内容について述べる。

### 2.1.1　量子ビット

既存のノイマン型アーキテクチャで用いられる一般的な古典ビットは 0 もしくは 1 のどちらかの状態をとる。これは、電気信号によって一定の電圧を超えた場合は 1 を、それ以下の電圧の場合は 0 を取る、と定義しているためである。一方で、量子コンピューティングは量子の性質を用いる。量子の重要な性質として、重ね合わせがある。重ね合わせとは、状態ベクトルの線形結合で表わせ、複数の状態ベクトルを用いることで、新たな重ね合わせの状態が作られる。ベクトルの重ね合わせ状態を持つ性質から、量子の物理状態は確率的に変化し、1 量子ビット (qubit) は以下の数式で表される。

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.1}$$

$\alpha$ および $\beta$ は複素数である。ただし、ここでは $|0\rangle$ と $|1\rangle$ を以下の通りに定義する。

$$|0\rangle \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{2.2}$$

$$|1\rangle \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.3}$$

$|\psi\rangle$ とは通称“ ケット (ket) ”と呼ばれる状態ベクトルの表記である。$|\psi\rangle$ のノルムは必ず 1 であり、如何なるユニタリゲートを通した場合でもその大きさは変化しない。つまり、$|\alpha^2| + |\beta^2| = 1$ である。ここで、よく使われるケットの例として 2 つのよく用いられる量子状態を示す。(式 2.4, 式 2.5)

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{2.4}$$

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{2.5}$$

これら 2 つの状態は、量子状態の重ね合わせを表現する際に、頻繁に用いられる。

### 2.1.2 量子エンタングルメント

複数の系が存在するとき、それぞれの系の状態を決定する確率はそれぞれ独立、もしくは従属する関係を取る。量子力学において、複数の量子状態を決定する確率が独立せず、相互に影響を及ぼす場合がある。この状態をエンタングルメントと呼ぶ。また、エンタングルメントの状態にあることを、エンタングルしていると言う。エンタングルメントは量子コンピューティングにも影響し、複数の量子の状態決定に関与する。なお、1 つの量子が複数の量子とエンタングルすることも可能である。
量子がエンタングルしている例として、式 2.6、2.7 を示す。式 2.6 はエンタングルしていない状態、式 2.7 はエンタングルしている状態である。ここでは、量子ビット A、量子ビット B を扱う。

$$|\psi\rangle = \frac{1}{2}(|0_A 0_B\rangle + |0_A 1_B\rangle + |1_A 0_B\rangle + |1_A 1_B\rangle) \tag{2.6}$$

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|0_A 0_B\rangle + |1_A 1_B\rangle) \tag{2.7}$$

式 2.6 では、$|0_A 0_B\rangle$ から $|1_A 1_B\rangle$ までのそれぞれの状態の確率が 25 パーセントづつであり、これは量子ビット A と量子ビット B が独立した状態である。一方、式 2.7 では、量子状態は $|0_A 0_B\rangle$ と $|1_A 1_B\rangle$ に限定されており、それぞれの確率は 50 パーセントである。この場合、量子ビット A の状態が 0 であるとわかれば、量子ビット B の状態もまた 0 であると推定できる。同様に、量子ビット A の状態が 1 であれば、量子ビット B の状態は 1 であると推定できる。このように、複数の状態のうち、取る値が偏ることで、一方の値を特定することでもう一方の値を特定することが可能となる。
ここで、量子ビットの数と、とりうる量子状態の数の関係について述べる。一般に、n 個の量子ビットがとりうる量子状態の数は $2^n$ 通りある。これは、古典ビットでも取り得る状態の数は同じである。しかし、量子状態の場合はこれらの $2^n$ の状態を同時に扱えるため、1 度に行える計算量が指数関数的に増加する。式 2.8 は、n ビットの量子ビットを扱う場合の量子状態の記述である。

$$|\psi\rangle = \begin{pmatrix} a_0|00\cdots000\rangle \\ a_1|00\cdots001\rangle \\ a_2|00\cdots010\rangle \\ \vdots \\ a_{2^n-2}|11\cdots110\rangle \\ a_{2^n-1}|11\cdots111\rangle \end{pmatrix} \tag{2.8}$$

### 2.1.3 量子ゲート

古典コンピューティングの物理演算には論理回路の組み合わせ回路が用いられるように、量子の演算にも量子ゲートが用いられる。図 2.1 は量子コンピュータ上で実現する、8 ビットの可逆計算可能な全加算器の回路 [8] である。この量子回路は、Cuccaro-Draper-Kutin-Moulton(CDKM08）[9] と呼ばれる。ここで、可逆計算とは、常に直前と直後の状態が一意に定まる計算であり、大量演算による CPU からの高熱発生を阻止する重要な計算である。

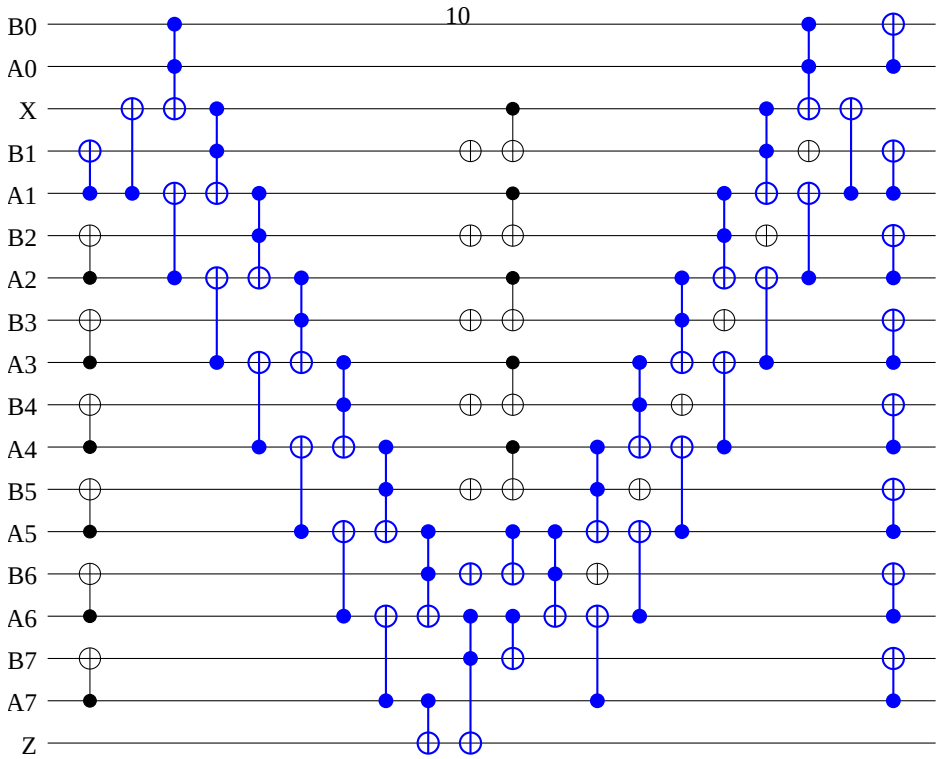


図 2.1: 量子回路の例：8 ビットの全加算回路

この量子回路図は左から右に時間が経過していく。各時間はステップで区切られ、1 ステップの中で複数の量子ゲートが実行される。図の最上段の 10 という数字は 10 番目のステップを意味する。
一番左の文字列 $B0, A0, X...$ は変数名であり、それぞれに引かれた横線の上にはそれぞれ各量子ビットが受けるユニタリ変換の処理内容が記述される。これらの変数が物理的に利用する量子ビットの数に相当する。図 2.1 では、$A0$〜$A7$、$B0$〜$B7$、$X$、$Z$ の合計 18 変数を用いる。
また、各量子ビットは各量子ゲートを通過する毎にユニタリ変換を受ける際に、異なる量子ビット同士であれば、1 ステップ中で複数の量子ゲートを扱える。各量子ゲートの意味については、第 2.2 節で詳しく述べる。

### 2.1.4　量子測定

量子ビットは、測定すると状態を一意に決定される性質を持つ。これは、複数の量子状態を持っている量子の状態を 1 つに決定することであり、量子状態は測定操作によって壊れてしまう. ここで、測定された量子ビットにおいて、$|0\rangle$ の係数を二乗した値が $|0\rangle$ が検出される確率、$|1\rangle$ の係数を二乗した値が $|1\rangle$ が検出される確率である.

量子ゲートでの表現には、以下に示す図 2.2 の Measure 回路が用いられる。

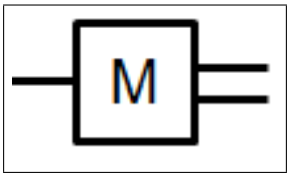


図 2.2: 量子測定回路

後ろの二重線は、古典ビットに変換されたことを意味する。

## 2.2　量子回路

ノイマン型コンピュータにも AND ゲートや NOT ゲートをはじめとする論理回路があるように、量子コンピューティングにも量子回路と呼ばれる独自の回路が存在する。これらの回路は、全てユニタリ行列によって表現可能である。以下に、量子回路のうち、本研究で扱う量子回路を述べる。

### 2.2.1　パウリ行列

パウリ行列は X、Y、Z ゲートの 3 種類あり、式 2.9、2.10、2.11 で表現される。それぞれ、X 軸方向、Y 軸方向、Z 軸方向の 180 度回転に相当する。

$$X = \begin{pmatrix} 0 & 1 \\ 1 & -0 \end{pmatrix} \tag{2.9}$$

$$Y = \begin{pmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{pmatrix} \tag{2.10}$$

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.11}$$

これらは、ユニタリー行列であり、かつエルミート行列の性質も持ち合わせている。また、パウリ行列をはじめとする多くの量子ゲートは2つの同じ量子ゲートを直列に連結させるとIゲート(何も操作を行わない量子ゲート)になる。Iゲートを式2.12に示す。

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.12}$$

### 2.2.2 Hadamard ゲート

図2.3はHadamardゲートの回路である。

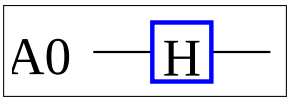


図 2.3: Hadamard ゲート

行列は、以下の式2.13のように表される。(以降のゲートについても同様)

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.13}$$

Hadamardゲートにより、$|0\rangle$ や $|1\rangle$ から重ね合わせ状態が生成される。(式2.14, 式2.15)

$$|+\rangle = H|0\rangle \tag{2.14}$$

$$|-\rangle = H|1\rangle \tag{2.15}$$

### 2.2.3 T ゲート

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} \tag{2.16}$$

図2.4はTゲート、図2.5は$T^\dagger$ (TD)ゲートと呼ばれる。Tゲートが $\frac{\pi}{4}$ の回転であるのに対し、$T^\dagger$ (TD)ゲートは $-\frac{\pi}{4}$ の回転、すなわち逆回転を意味する。なお、Tゲートは歴史的経緯から $\frac{\pi}{8}$ ゲートと呼ばれることがある。なお、$\dagger$ の表記は他の量子ゲートにも適用可能であり、$\dagger$ のつく量子ゲートは、通常の量子ゲートに対して随伴行列の関係にあたる。

これらは、1量子ビットの回転操作として用いられる。なお、1量子ビットのZ軸方向への回転は、一般に、式2.17として表現される。ここで、$\theta$ は量子ビットのZ軸方向の回

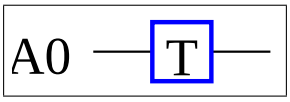


図 2.4: T ゲート

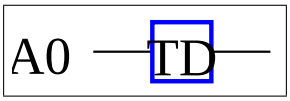


図 2.5: T† ゲート

転角度に相当する。

$$R_z(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix} \tag{2.17}$$

### 2.2.4 S ゲート

図 2.4 は S ゲート、もしくは位相ゲートと呼ばれ、$\frac{\pi}{2}$ 回転を意味する。なお、T ゲートを 2 つ直列につなげると S ゲートと等価になる。つまり、$S = T^2$ である。また、パウリ行列を用いて $Z = S^2$ という式も成り立つ。

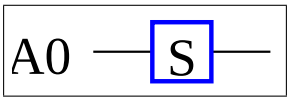


図 2.6: S ゲート

### 2.2.5 CNOT ゲート

図 2.7 は CNOT ゲートの回路である。

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.18}$$

このゲートは、２つの量子ビットを扱う。片方が Control ビットと呼ばれ、もう片方のビットの制御を行う。もう一方は Target ビットと呼ばれ、Control ビットが 1 であれば状態が反転し、0 であれば状態が維持される。（表 2.1）

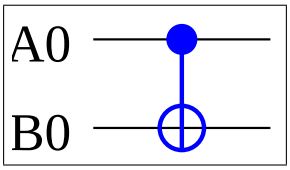


図 2.7: CNOT ゲート

表 2.1: CNOT ゲート真偽表

| Input $\|AB\rangle$ | Output $\|A\rangle\|B\oplus A\rangle$ |
|---|---|
| $\|00\rangle$ | $\|00\rangle$ |
| $\|01\rangle$ | $\|01\rangle$ |
| $\|10\rangle$ | $\|11\rangle$ |
| $\|11\rangle$ | $\|10\rangle$ |

CNOT ゲートに関わる 2 つの量子ビットは必ずしも隣接していなくても構わない。“ 隣接している ”とは量子ビットが物理的、または論理的な距離が 1、すなわち最小距離単位である場合である。逆に、“ 隣接していない ”とは距離が 2 以上の場合であり、例えば距離が 2 である 2 隣接の CNOT ゲートであれば、2 neighbor-CNOT と呼ぶ。また、距離が N の場合は N neighbor-CNOT である。
図 2.8 が 2 量子ビット離れた場合、図 2.9 が 4 量子ビット離れた場合の CNOT ゲートの回路である。

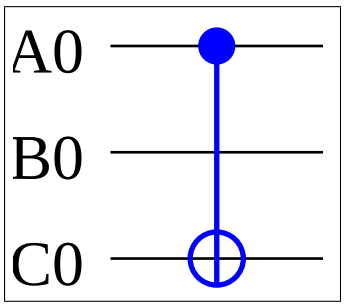


図 2.8: 非隣接ゲートの例
（2 量子ビット離れた CNOT ゲート）

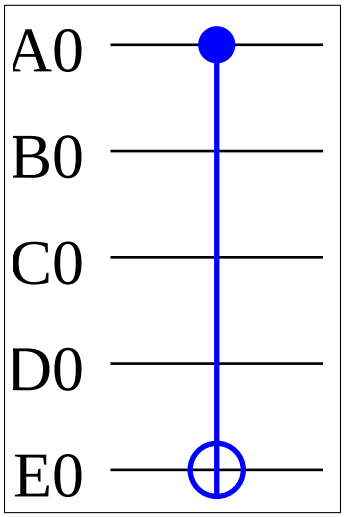


図 2.9: 非隣接ゲートの出力例
（4 量子ビット離れた CNOT ゲート）

このような場合、第 2.2.7 節で述べる SWAP ゲートを用いることで、既存の隣接状態にある CNOT ゲートを使える。(図 2.10,2.11) これは、ユニバーサル回路を構成する上で重要である。

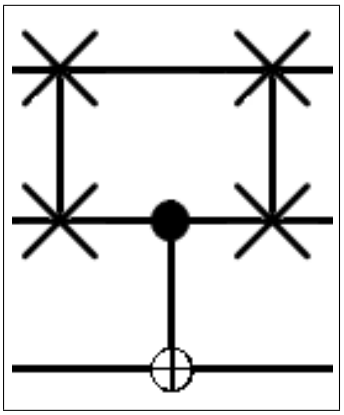

図 2.10: 2 量子ビット離れた CNOT ゲート: SWAP ゲートを活用したモデル

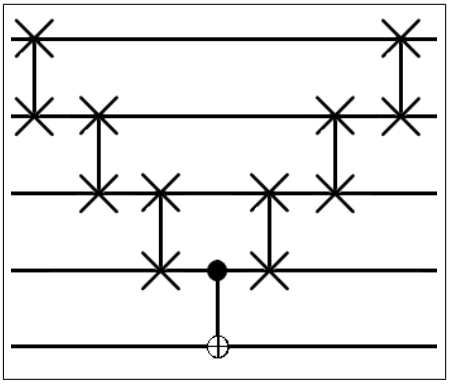

図 2.11: 4 量子ビット離れた CNOT ゲート: SWAP ゲートを活用したモデル

### 2.2.6 CPhase ゲート

$$CPhase = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \tag{2.19}$$

CPhase ゲートは、Target ビットに対して Z ゲートの操作を行う行列である。CNOT ゲートの場合は Target ビットに対して X ゲートの操作を行う。
また、CPhase ゲートは CNOT ゲートと Hadamard ゲートによって記述可能であり、図 2.13 は図 2.12 と等価回路である。等価回路とは、機能が等しい回路であり、どちらの回路を用いても等しい結果を得ることができる回路のことを指す。

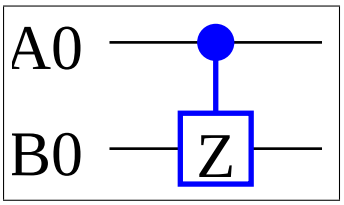


図 2.12: CPhase ゲート

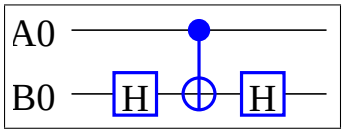


図 2.13: CPhase ゲート (CNOT ゲートによる等価回路)

また、CPhase ゲートは上下の区別がない。これは、CPhase ゲートがどちらを Control ビット、Target ビットにしても全く同じ役割を果たすためである。

### 2.2.7 SWAP ゲート

SWAP ゲートは、隣接するビットにおいてだけ使え、上下の量子ビットの状態を入れ替える。

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.20}$$

図 2.14 は、SWAP ゲートの回路図である。また、図 2.15 は SWAP ゲートと等価な回路であり、CNOT ゲートのみで構成されたものである。CNOT ゲートを 3 つ並べることで SWAP ゲートは実現可能である。このゲートにより、非隣接型の回路を隣接型の回路に変換することできる。

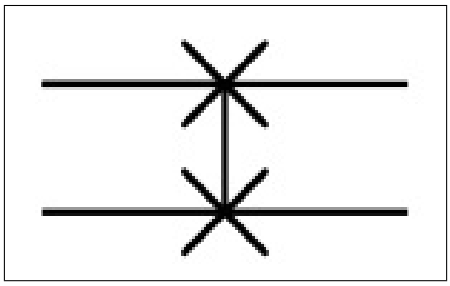

図 2.14: SWAP ゲート

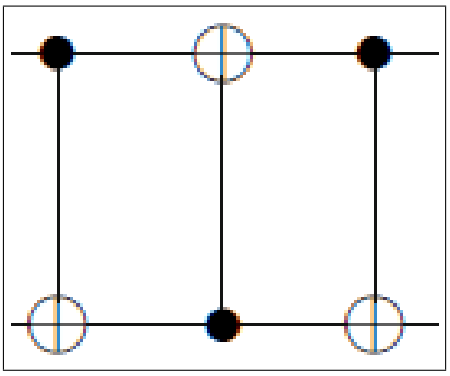

図 2.15: CNOT ゲートのみで表現された SWAP ゲート

### 2.2.8 Toffoli ゲート

Tofolli ゲートとは、今までのゲートとは異なり、3 量子ビットを対象として操作を行う。CCNOT ゲートとも呼ばれる。Control ビットが 2 つ存在し、1 つの Target ビットを制御する。つまり、 2 つの Control ビットが 1 である場合に限り、Target ビットの状態が反転する。
表 2.2 に、Toffoli ゲートの真偽表を記す。ここで、量子ビット A,B が Controll ビットであり、量子ビット C が Target ビットである。

なお、Toffoli ゲートを行列で表現する場合、3 量子ビットが関わっているため、$2^3 = 8$

表 2.2: Toffoli ゲート真偽表

| Input $\|ABC\rangle$ | Output $\|A\rangle\|C\oplus(A\wedge B)\rangle$ |
|---|---|
| $\|000\rangle$ | $\|000\rangle$ |
| $\|001\rangle$ | $\|001\rangle$ |
| $\|010\rangle$ | $\|011\rangle$ |
| $\|011\rangle$ | $\|010\rangle$ |
| $\|100\rangle$ | $\|100\rangle$ |
| $\|101\rangle$ | $\|101\rangle$ |
| $\|110\rangle$ | $\|110\rangle$ |
| $\|111\rangle$ | $\|111\rangle$ |

量子状態、すなわち 8 行 8 列の正方行列によって表現される。（式 2.21）

$$\text{Toffoli} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.21}$$

量子回路図を以下の図 2.16 に示す。

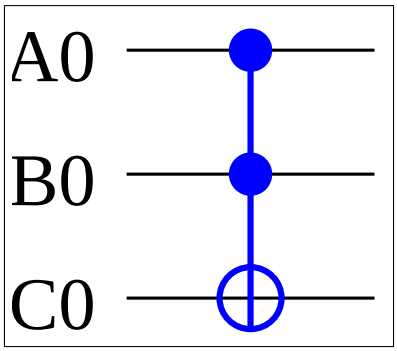


図 2.16: Toffoli ゲート

Toffoli ゲートの等価回路を図 2.17 に示す。

このゲートは 14 ステップから構成され、2 つの Hadamard ゲート、6 つの CNOT ゲート、4 つの T ゲート、4 つの T† ゲートから構成される。

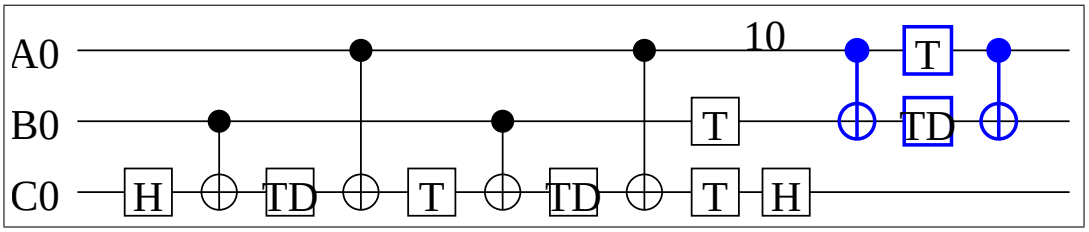


図 2.17: Toffoli ゲートの等価回路例

## 2.3 量子回路の機能的完全性

古典回路において、NAND ゲート (図 2.18) は機能的完全性を備えている。

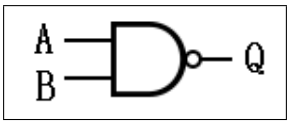


図 2.18: NAND ゲート

NAND ゲートは NOT ゲートと AND ゲートを合わせた形をしており、真偽表を表 2.3 に示す。AND ゲートは入力情報 A、B が 1 であった場合に出力情報が 1 となるのに対し、NAND ゲートは双方が 1 となった場合のみ出力結果が 0、それ以外の場合が 1 となる。

表 2.3: NAND ゲート真偽表

| InputA | InputB | Output(A NAND B) |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

機能的完全性とは、あるゲートの集合 (組) がそのゲートの属する集合の他の全てのゲートの機能を満たすことが可能な集合のうち、最小のゲート集合を指す。これは、より少ないゲートの実装によって全ての機能を賄う上で重要である。例えば古典論理回路では、NAND ゲートによってその他の全てのゲート (AND, OR, NOT など) の機能を満たせる。

量子回路においても機能的完全性を満たすいくつかの組み合わせがある。最初の例は、Toffoli ゲートによる方法である。

- Toffoli ゲート

- 1 量子ビットの回転

Toffoli ゲートは、古典回路における任意のユニタリー演算を実行できる。量子回路の場合、これに 1 量子ビットの回転を加えることで任意の演算を可能とする。
ここで述べた Toffoli ゲートはいくつかの等価な量子回路に分割できる。図 2.17 に必要な、以下の 4 つの量子ゲートの組み合わせによって最小の回路セットを実現できる。

- Hadamard
- CPhase
- CNOT
- T

更に、Hadamard ゲートは X ゲートと Z ゲートで表現可能である ($H = \frac{1}{\sqrt{2}}(X + Z)$) ことを用いる。Hadamard ゲートと T ゲートを合わせて 1 量子ビットの回転を表現し、Cphase を CNOT と Hadamard ゲートの等価回路に分解することで、この 4 組を以下の 2 つの演算ゲートで表現できる。

- 1 量子ビットの回転
- CNOT ゲート

以降、機能的完全性のことを“ ユニバーサル ”、または機能的完全性を備えた回路の組のことを“ ユニバーサルセット ”と呼ぶ。ユニバーサルセットの問題点として、計算量の多さが挙げられる。上記に挙げた 3 種類の組み合わせは、全てのゲートを表現できる反面、最適化に限界が生じる。速い量子アルゴリズムを行うには、ユニバーサルセットとは異なるアプローチが必要である。

## 2.4 測定ベース量子計算

測定ベース量子計算とは、MBQC（Measurement Based Quantum Computation）[10] のことであり、量子コンピュータのアーキテクチャの 1 つである。多数の格子状に配置されたエンタングルされた量子ビットを扱い、それらを定められたアルゴリズムで測定していくことで計算を進めていく手法である。正確には MBQC の内部には様々なアーキテクチャが存在し、以降で述べる Brickwork state もまた MBQC の一種であるが、通常、MBQC は square lattice と呼ばれるアーキテクチャを指す。したがって、本研究では、MBQC アーキテクチャと呼ばれた場合、square lattice アーキテクチャを指す。
図 2.19 は、MBQC の計算の一例である。
これは、MBQC 上で CNOT ゲートを実行する場合のアーキテクチャモデル [11] である。まず、入力情報として量子ビットと実行する関数をセットする。量子ビットは光の場合は偏光であり、実行する関数は実行する内容によって異なってくる。それによって、どの量

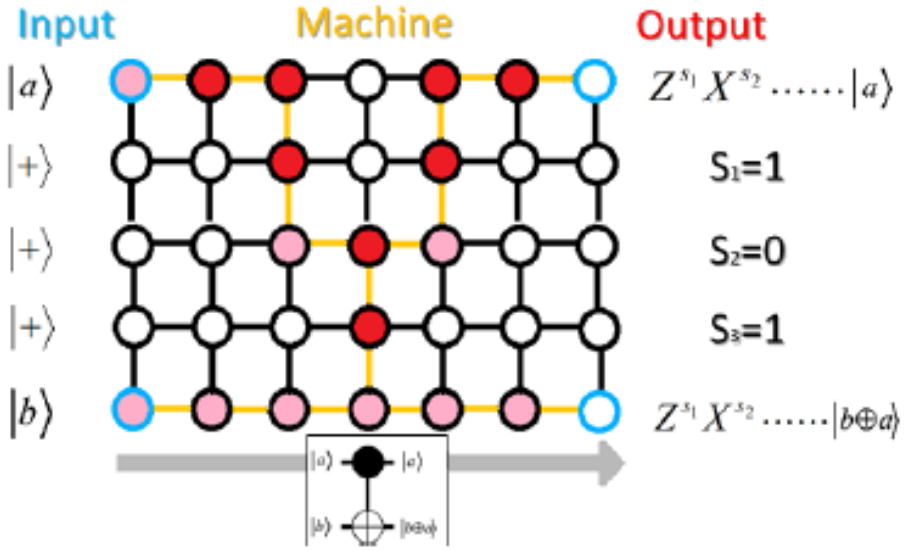


図 2.19: 測定ベース量子計算の流れ

子ビットのエンタングルが扱われ、そのように測定するかが定められる。図 2.19 の場合、赤い丸は、X 軸方向での測定、ピンク色の丸は Y 軸方向での測定を意味する。そして、黄色の線は格子状のエンタングルメントのうち、計算として活用する部分を表す。
この計算もまた、左の列から右の列へ計算を行う。エンタングルメントによる量子ビット間での情報伝搬が終わった後に順番に測定を行っていき、出力結果である古典ビットを順次入手する。これは、第 2.1.4 節で述べた量子測定の効果を用いている。
エンタングルメントされた量子ビットの一部を測定した後、残った量子ビットの推定に入る。推定の際には出力された古典ビットを活用し、量子ビットとして残っている部分に適切なオペレーションを施すことで、得たい量子状態、すなわち演算結果を得られる。

最も単純な測定ベース量子計算の例として、図 2.20 の量子テレポーテーションを行う回路 [7] を挙げる。この回路は、一番上の量子ビットから一番下の量子ビットへと情報 $|\Psi\rangle$ を送信する。

この回路の意味としては、まず、入力された $|\Psi\rangle$ は、$|+\rangle$ と Cphase ゲートによって相互作用を行う。そして、上 2 つの量子ビットに適切な回転を行った後、測定を行う。測定を行うことにより、量子ビットは古典ビットに変換される。これらの古典ビットの情報は、一番下の量子ビットに付加された論理的な行列を取り除くために用いられる。ここで、$S_1$, $S_2$ は 1 であれば Z ゲート、X ゲートによる行列が適用される。このように、量子ビットから量子ビットへの情報の転送が行われる。
MBQC において、いくつかのモデルが提唱されているが、本研究においてはピッチが 4 の MBQC モデル [11] を採用する。ピッチは、通常の量子回路と比較した場合に、どれだけ物理的に量子ビットが離れているかを表す。通常の回路であれば、量子ビットは隣り合って

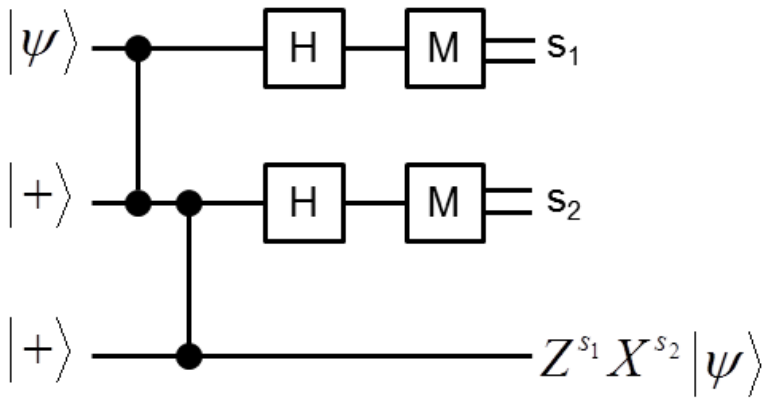


図 2.20: 量子回路の例：3qubit の量子テレポーテーション

いる前提があり、ピッチは 1 であるが、本研究において活用する MBQC は隣接する論理量子ビット同士の間に 3 つの量子ビットを必ず挟む構造となっているため、ピッチは 4 である。ピッチが 4 のモデルは、既に多くの量子回路が実装されているため [10]、ユニバーサル計算に活かしやすい。

## 2.5 Brickwork state

Brickwork state[12] とは、量子計算アーキテクチャのうちの 1 つであり、図 2.21 に示したパーツを基本としており、前節の測定ベース量子計算と関連する。

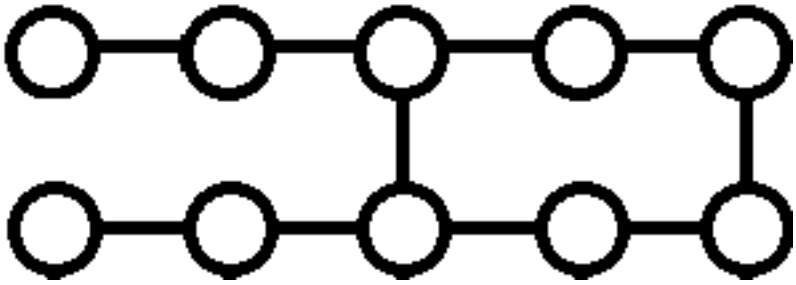

図 2.21: Brickwork state の 1 パーツ

1 つのパーツは 10 個の量子ビットから構成され、図 2.21 に定められた形で必ずエンタングルメントを行っている。1 つの関数（Hadamard Gate, CNOT Gate）につき、1 パーツがその関数の役割を担う。この 1 パーツは、測定ベース量子計算と同様に左から右に関数として指定された角度で測定を行うことで演算を行う。例として、図 2.22 に Hadamard

ゲートの Brickwork 形式の回路を示す。

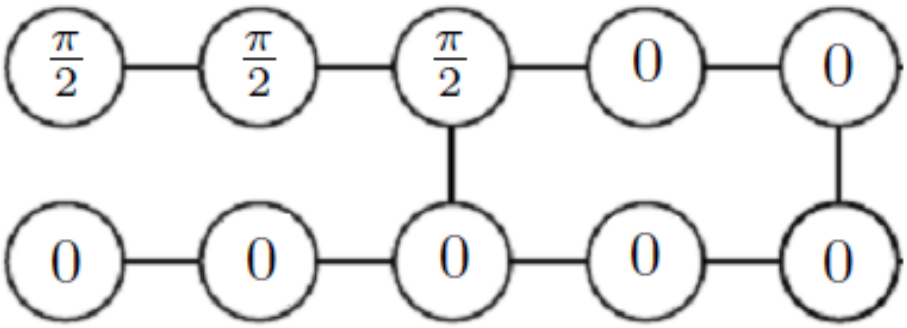


図 2.22: Brickwork state による Hadamard ゲート

図中の $\pi/2$ とは、$\pi/2$ 回転させた後に量子測定を行う意味である。0 は何も回転を行わずに量子測定を行う。
MBQC と異なり、Brickwork state では必ず Z 軸方向で量子測定を行う。これは、前段階の回転角度 (先ほどの 0、$\pi/2$ の部分) の部分で量子測定による矯正を必要としないように回転角度を定めるためである。Brickwork state の利用者は測定前に量子ビットにどれだけの回転をかけるか、何の関数を使うのか、といった部分だけを考えればよい。これは、第 2.7 節で述べる秘密量子計算を実現する上で重要な性質である。

Brickwork state の 1 パーツを多数つなげたものが図 2.23 である。

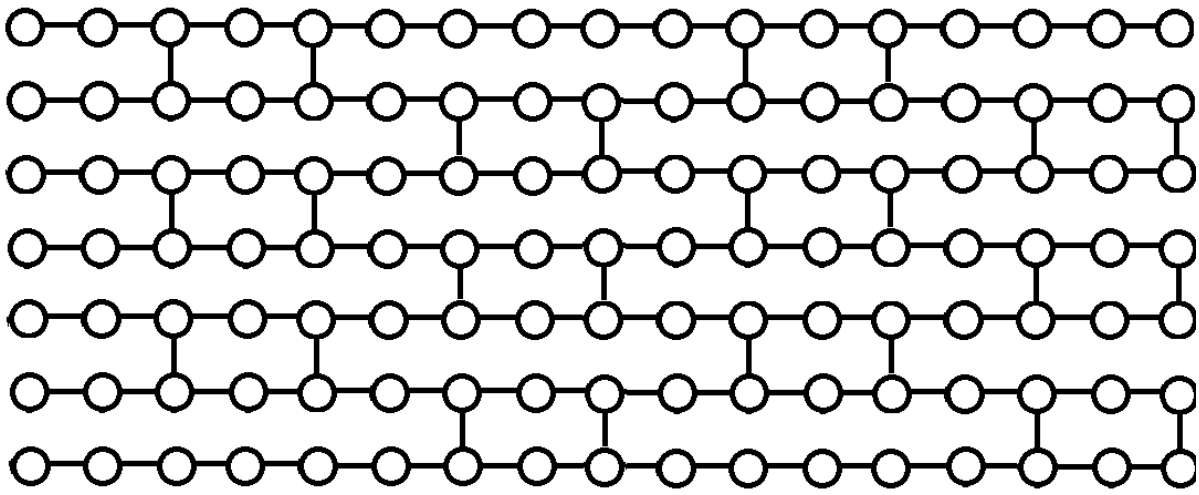

図 2.23: 大規模な Brickwork state アーキテクチャ

図 2.23 の通り、Brickwork state はパーツが交互に並ぶ構造を取る。このため、例えば 2 量子ビットの相互作用を同じ論理ビット上で 2 回続けては行えない。例えば、SWAP ゲート（第 2.2.7 節）のような回路を実行する場合、1 ブロック区間目で CNOT ゲートを実行し終えた後は、次に 3 ブロック区間目でリバース CNOT ゲート (CNOT ゲートの Control ビットと Target ビットが反転したもの) を実行し、その次は 5 ブロック区間目で CNOT ゲートを実行することになる。ただし、MBQC とは異なり、ピッチが 1 であるため、縦方向を論理ビットと同じ形で扱える。

## 2.6 秘密計算

秘密計算とは、入力・演算・出力を全て暗号化した状態で計算を行うことができるクライアント・サーバ方式の演算である。図 2.24 は秘密計算の処理の流れである。
まず、クライアントは演算に用いる入力“ x ”および演算内容“ F ”（アルゴリズム）を

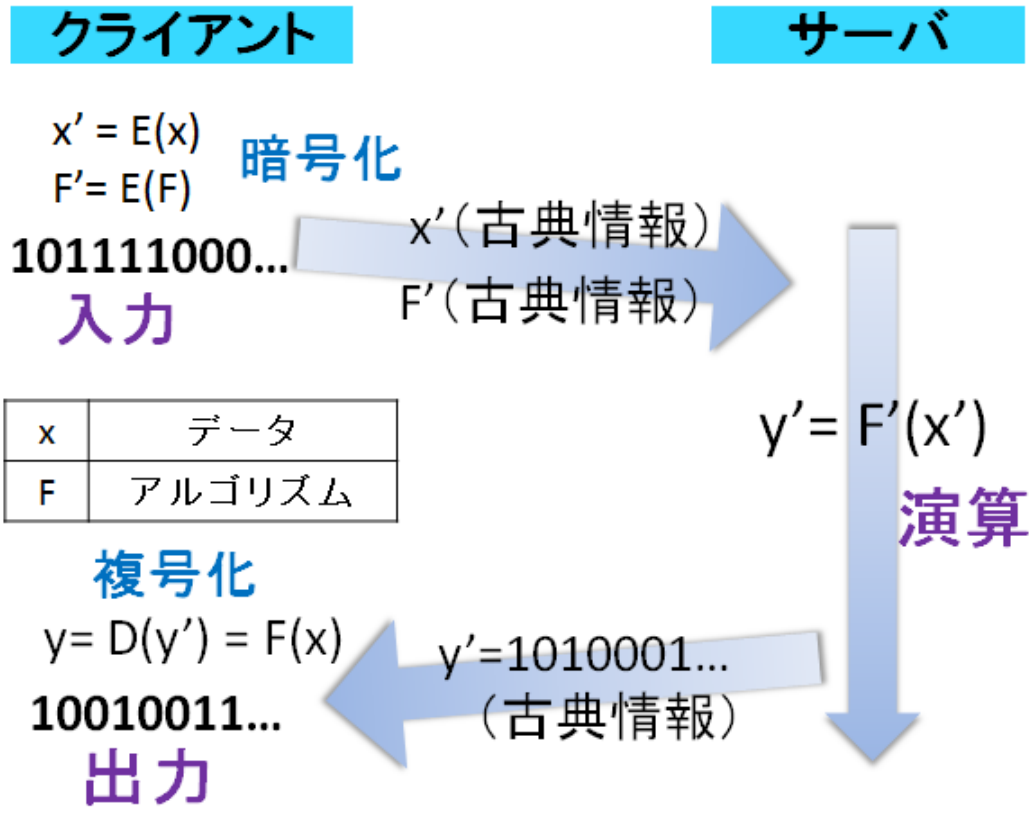


図 2.24: 秘密計算の処理手順

用意する。次に、入力 x とアルゴリズム F をそれぞれ暗号化関数“ E ”によって暗号化する。暗号化したデータ $x'$ と $F'$ をサーバに送信する。

次に、サーバはクライアントから $x'$ と $F'$ を受け取り、そのまま演算を行う。これにより、出力された結果 $y'$ をサーバはクライアントに返す。クライアントは、サーバから得た $y'$ を元にして、関数 $E()$ と対になっている復号関数 $D()$ を使って、$y'$ を復号し、本来計算したい $y = F(x)$ の結果である y を手に入れられる。
古典コンピューティングで実現される秘密計算の一例として、Gentry の暗号 [1] がある。Gentry の暗号は、完全準同型暗号方式 [13] を採用しており、公開鍵方式をベースとしている。この通信方法の最大の利点は、サーバ内に盗聴者がいた場合でもサーバ自身が本来のデータ・アルゴリズムについて何も知らないため、内容を傍受されることがないことである。
しかしながら、本来の計算量に比べて、大きなオーバーヘッドが発生する。

## 2.7 秘密量子計算

秘密量子計算 (Blind Quantum Computation:BQC) は [14]、第 2.6 節の秘密計算と第 2.4 節の測定ベース量子計算を組み合わせた技術である。この処理手順を図 2.25 に示す。

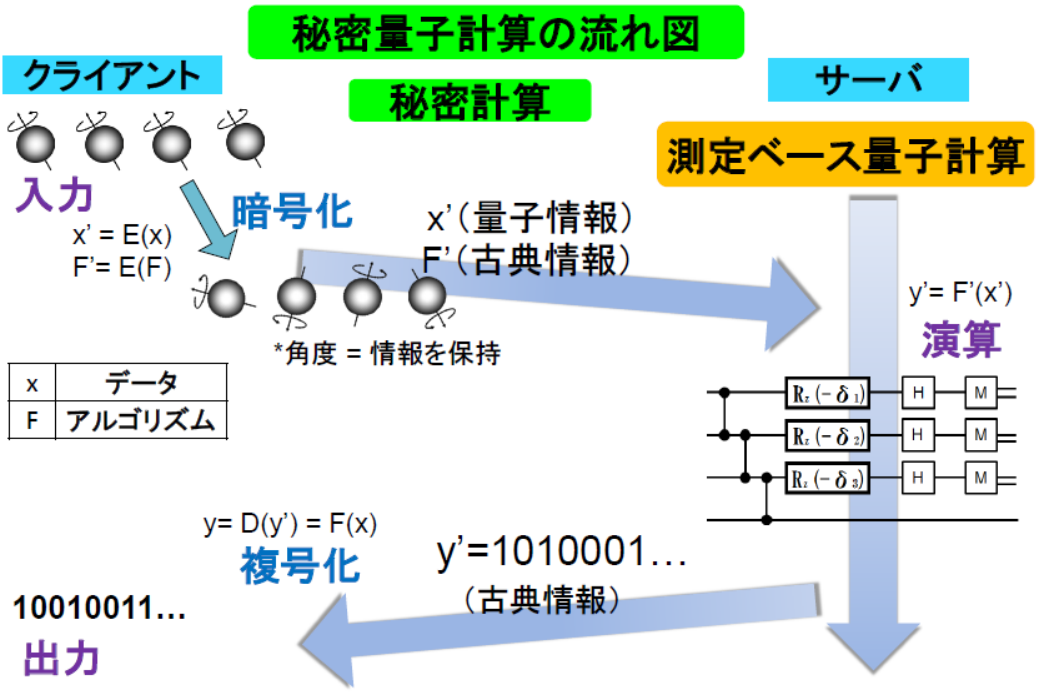


図 2.25: 秘密量子計算の処理手順

まず、クライアントは演算に用いる量子ビット“ x ”(入力)、および演算内容“ F ”(アルゴリズム）を用意する。次に、入力 x とアルゴリズム F をそれぞれ暗号化関数“ E ”によって暗号化する。暗号化したデータ $x'$ と $F'$ をサーバに送信する。この時、$x'$ は量子ビットであり、ここでは量子とは光子のことを指すので、光の偏光角度を用いて入力情報の管

理、暗号化を行う。アルゴリズムはプログラム内容であり、そのうちの測定角度の部分を暗号化した後に、古典通信でサーバへ送信する。
次に、サーバはクライアントから $x'$ と $F'$ を受け取り、そのまま演算を行う。演算には測定ベース量子計算の一種である Brickwork state を用いる。Brickwork state は暗号化された入力量子ビットの情報、およびプログラムを受け取る。ここで、プログラムの中身は測定する角度の部分が暗号化されており、Brickwork tate の演算は測定する角度そのものが関数であるため、関数の暗号化は実現される。これにより、出力された結果 $y'$ をサーバはクライアントに返す。
クライアントは、サーバから得た $y'$ を元にして、関数 $E()$ と対になっている復号関数 $D()$ を使うことで、$y'$ を復号することで、本来計算したい $y = F(x)$ の結果である y を手に入れられる。
以上の秘密量子計算の利点は、秘密計算と比べた場合に、比較的実装が行いやすい、計算量が減るということである。

## 2.8 コンパイラの原理

本研究では、量子回路のコンパイラ [15] の拡張を行う。これは、既存のコンパイラ技術を基盤としているため、コンパイラの原理について述べる。コンパイラは、プログラミング言語で書かれたプログラムのソースコードを、目的のコードに翻訳・変換するためのプログラムである。目的のコードは機械語をはじめとする、実行環境に依存したコードになる。コンパイラによる翻訳・変換の過程のことをコンパイル、またはコンパイルする、と呼ぶ。本研究では、数多くのコンパイラのうち、ハードウェア記述言語（HDL）を対象とする。図 2.26 に、古典コンパイラの処理概要を示す。

1. **原始プログラム**
   入力に使用されるプログラムのことである。通常はプログラムから行単位で読み込み、そして文字単位に分割する。分割した文字は次の字句解析に活かされる。

2. **字句解析**
   字句解析は、字句解析器によってソースコードの文字の並びをトークンと呼ばれる最小の意味内容に分割する。各トークンはそれぞれ意味を持ち、字句解析器の語彙素によって分類される。例えば、$sum = 2 + 4$ であれば、$sum$ は VARIABLE、2, 4 は NUMBER,+は ADD, = は ASSIGN、といった具合である。
   分類された字句は、次の構文解析に活かされる。

3. **構文解析**
   構文解析は、字句解析を元にして単語から文がどのような構成を成すか調べる。また、構文解析器によってトークンの並びが文法規則に従っているかどうかを判定する。判定には、構文木と呼ばれる文法がどの規則に対応するかを判定し、プログラムの構造を示す構造体を用いる。


図 2.26: 一般的なコンパイラの論理的構造

先ほどの $sum = 2 + 4$ であれば、ここでは 2 と 4 を足し合わせ、結果を変数 $sum$ に代入する、という意味を得る。

4. **中間コード生成**
   目的コードを得るための前段階として、中間コードを生成する。中間コードは構文木を元にした内容であるが、目的コードに近づけるために、実行時に実行される順に命令を並べ替えることがある。

5. **最適化**
   最適化とは、目的プログラムを実行する時に効率よく実行するために最適なものに近づける処理である。この処理は、構文木、生成されたコードの段階で行われることもある。例えば、プログラムの順序を入れ替えたり、活用可能な同一のリソースを使うことでメモリを節約するといった内容である。

6. **コード生成**
   中間コードを元にして、目的コードを組み立てる。生成されたコードは通常ファイルに出力される。

以上が、コンパイラの論理的構造である。コンパイラは、オブジェクトコードを生成しながら実行するインタプリタ型言語と比べると高速に実行できる点で優れている。
量子回路もまた、このコンパイラの論理的構造に従ってコンパイルすることで、目的の量子回路を生成することが可能である。

# 第3章　秘密量子計算の設計

## 3.1　秘密量子計算における Brickwork state の導入

### 3.1.1　概要

本研究の秘密量子計算は測定ベース量子計算の一種である Brickwrok state を用いている。測定ベース量子計算の様々なアーキテクチャの中で秘密量子計算が Brickwork state を採用しているのは、暗号化する際に測定する前処理の回転角度のみを暗号対象とすればよい点である。これは、Brickworkstate において重要なアルゴリズム情報はいくつの量子ビット平面を扱うのか、及び測定の前にかける偏光角度に統一されているためである。これは、ユニバーサルな回路を作る上でも役立つ。

### 3.1.2　秘密量子計算の行程

図 3.1 に、秘密量子計算の流れを示す。

まず、任意の高級言語を使い、量子回路を記述したプログラムを作成する。本研究では、AQUA 言語 [6](第 4.2 節) を用いる。ここで、ユニバーサル秘密量子計算の条件を満たすためには、そのプログラムもまた機能的完全性の条件を満たしている必要がある。
次に、測定ベース量子計算回路への変換である。本研究では、MBQC 回路として Brickwork state を用いる場合を考える。測定ベース量子計算において重要な内容は、測定する角度、どの量子ビット同士がエンタングルメントを行っているか、といった 2 種類の情報である。生成された 2 種類の情報から、関数 F とデータ X を生成する。Brickwork state では量子ビット同士が一定の法則に従ってエンタングルメントを行っているため、関数 F の内容としては、古典情報である測定角度（F）と、量子情報に変換する偏光（x）の 2 つを用いる。ただし、この段階では偏光 (x) は、プログラムから抽出されたものであるため、古典情報であり、暗号化を行った後に量子情報に反映させる。
この 2 つの情報を任意の復号可能な暗号化方式で暗号化し、F’、および x’ を生成する。F’ はこのままサーバに送信され、入力されたデータ x’ はクライアントが量子情報に内容を反映させて、サーバに送信する。これにより図 3.1 の通りの秘密量子計算の過程をプログラムで実現できる。
本研究では、これらの行程のうち、最初の部分である高級言語から、測定ベース量子計算モデルへの変換を扱う。図中の黄色い枠で囲まれた部分がそれに該当する。この変換の部分は、今後の演算が大規模化していくことを想定した場合、重要である。この秘密量子


図 3.1: 秘密量子計算プログラムの行程

計算は、最小の回路でも 5 量子ビット、大きなプログラムを組んだ場合数千個、数万個といった多数の量子ビットを扱う。本研究ではそれらの量子ビット数の最適化を扱う。

## 3.2 Brickwork state における SWAP ゲートの活用

Brickwork state に限らず、量子回路中で SWAP ゲートは重要な役割を果たす。以下にて、その理由、問題定義について言及する。

### 3.2.1 SWAP ゲートの用途

SWAP ゲートは、物理的に離れた量子ビットを論理的に隣接させ、少ない回路の実装でユニバーサル回路を利用できるようになる。これは、第 2.2.5 節で述べた CNOT ゲートに限られた話ではない。以下に示す図 3.2, 図 3.3 は Toffoli ゲートへと SWAP ゲートを適用し、Toffoli ゲートと SWAP ゲートから新しい回路を実現した例である。

ユニバーサルな回路を実現する上で、量子ビット間の相互作用に相当する量子ゲートの実装は必須である。一般的な量子回路を実現する上で、複数の量子ビットで相互作用を行

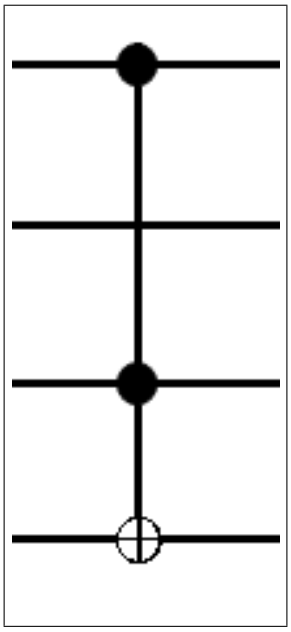

図 3.2: 非隣接型 CCNOT ゲートの一例

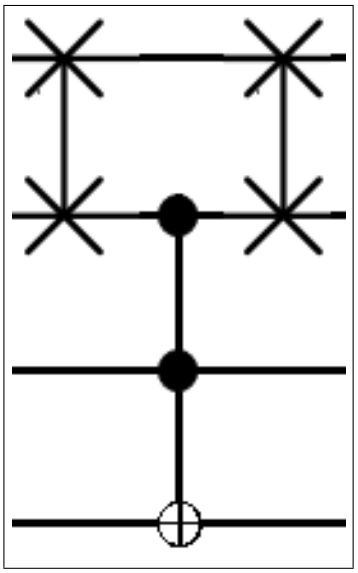

図 3.3: CCNOT ゲートへの SWAP ゲートの活用

う量子ゲートは頻繁に登場する。そして、そのゲートは必ずしも隣接しているとは限らない。このような場合に SWAP ゲートを実装し、他の量子ゲートに活用することは実装する量子ゲートの数を減らすことに大きく貢献する。これより、本研究では特に SWAP ゲートに着目した秘密量子計算の検証を行う。

### 3.2.2 Brickwork state における SWAP ゲートの問題点

図 3.4、3.5、3.6、3.7 は、MBQC モデルにおいて登場する SWAP ゲートの回路 [10][11] である。

MBQC 上で実装される SWAP ゲートには現在 2 通りのモデルが存在するが、どちらも機能に違いはなく、量子ビットの配置方法が異なるのみである。この回路を実現する場合、格子配列上に 4 行 4 列のスペースを必要とする。しかし、Brickwork state は図 2.21 による定められた回路の上のみを動作するため、枠に入らない変則的な回路を実行できない。そのため、Brickwork state に対応した新しい SWAP ゲートの実装が必要である。

## 3.3 研究目標

本研究では、Brickwork state 上にて SWAP ゲートを実装することにより、SWAP ゲートの有用性を検証する。SWAP ゲートを適用する回路は CNOT ゲートの非隣接モデルを用いる。

また、MBQC 回路、および Brickwork state の回路を効率よく生成するためのコンパイラ（トランスレータ）に実装も併せて行うことで、検証の効率化を図る。

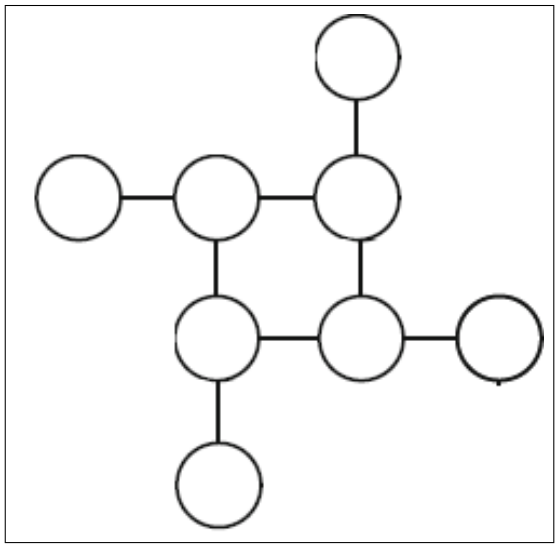

図 3.4: MBQC アーキテクチャにおける SWAP ゲートの実装 (1)

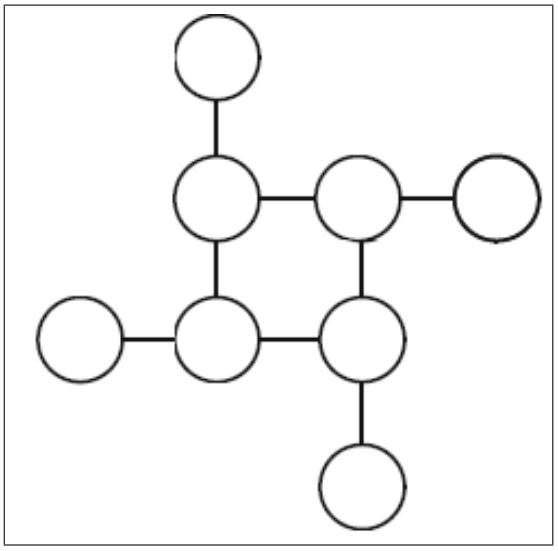

図 3.5: MBQC アーキテクチャにおける SWAP ゲートの実装 (2)

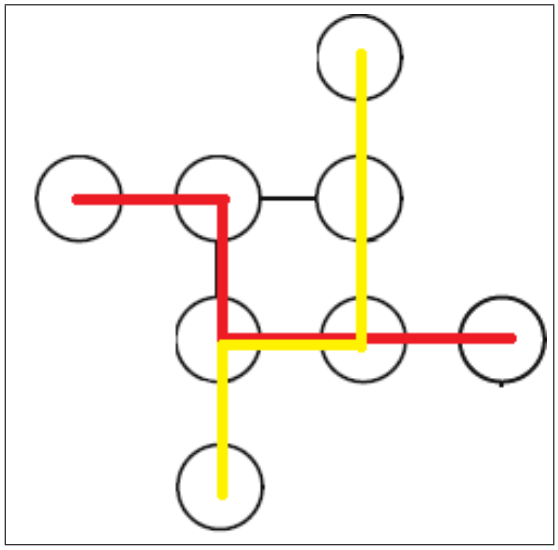

図 3.6: MBQC における SWAP ゲートのエンタングルメント (1)

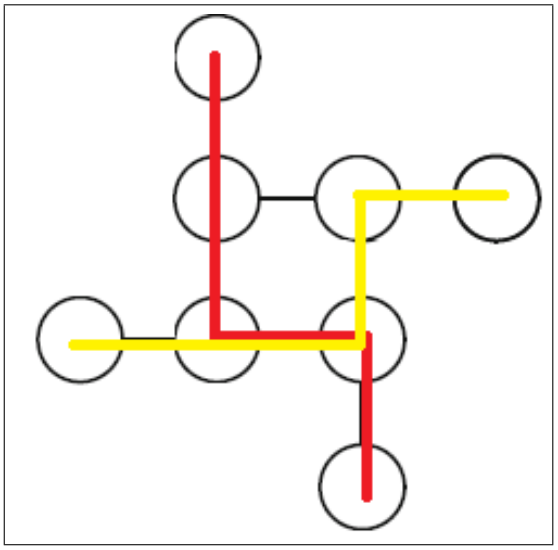

図 3.7: MBQC における SWAP ゲートのエンタングルメント (2)

# 第4章　実装

## 4.1　古典コンピュータ上で動作する量子回路トランスレータの設計

本研究では、古典コンパイラを応用することで、量子回路をあるアーキテクチャから別のアーキテクチャに変換するトランスレータの実装を行う。
図 4.1 は、本研究で実装する古典コンパイラの流れである。


図 4.1: 量子コンパイラの処理内容

この流れは、以降述べる MBQC トランスレータ、Brickwork トランスレータに応用される。既存のコンパイラ (第 2.8 節) との違いは、メモリ配置の部分が量子ビットにフォーカスしている点である。これは、各量子ビットの集まりがそれぞれ演算内容を持つためである。
字句解析、構文解析には次節 4.2 で述べる AQUA 言語を用いる。これによって生成された

中間コードは量子アーキテクチャに対応した、どの量子ビットにどの関数が関わるのか、といった量子回路にフォーカスした内容である。そして、最適化は量子ビットの数、実行する演算の量、ステップ数などに着目し、それらの内容を最適なものへと変える。

## 4.2 AQUA 言語

AQUA 言語 [6] とは、AQUA プログラム上で動作するプログラムのことである。ここでは、その特徴、および処理内容について述べる。

### 4.2.1 AQUA プログラムの実行

図 4.12 は、AQUA 言語を活用した AQUA プログラムの処理内容である。

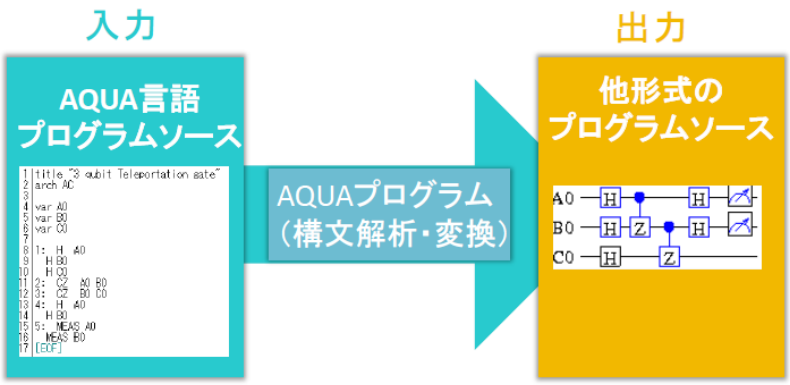


図 4.2: MBQC 変換コンパイラの処理内容

AQUA プログラムは AQUA 言語を読みこみ、構文解析を行い、指定された形式のファイルに変換する。AQUA プログラムには、コンパイラとしての機能が備わっており、ある量子回路を別の量子回路に変換したり、AQUA 言語によって記述された量子回路そのものの解析を行うことも可能である。
以下に、AQUA プログラムに含まれる主要なプログラムの内容について述べる。

1. **aqua.c**
   AQUA プログラムの中におけるメインプログラムである。入力されたファイルの形式 (拡張子) によって、以降の処理する内容をアーキテクチャに特化したものに定める。
   処理する内容は、図 fig:quantumcompiler の通りである。

2. **aquastructs.h**

AQUA プログラムにおける変数定義、中間コードを格納する構造体などが記述されている。本研究と関連する構造体について述べる。QuantumProgram 構造体は、

```
struct QuantumProgram {
  char *title;
  char *outfilename;
  struct TargetArchitecture *outarchp;
  struct TargetArchitecture *inarchp;
  struct gate *gatep;
  int gatecount;
  int TotalArgCount;
  int *CumulativeArgCountToTimeSlice;
  int *ArgCountInTimeSlice;
  int *CumulativeGateCountToTimeSlice;
  int *GateCountInTimeSlice;
  int circuitdepth;
  int progspace; /* storage allocated */
  struct var *variables;
  int numvars; /* in use */
  int varspace; /* storage allocated */
  struct storageloc *storage;
  int storagespace;
       .
       .
       .
};
```

図 4.3: aquastructs.h 内部の Quantumprogram 構造体の一部

中間コード生成の際に作られた中間コードに相当する。プログラムのタイトル名・アーキテクチャ名、扱う変数の数、量子ゲートなどといった目的コード生成に活用される情報が格納される。例えば、title はソースコード内部のタイトル、inarchp、outarchp はそれぞれ入力、出力されるアーキテクチャ形式、numvars はプログラムが扱う物理的な変数の合計である。
特に、gate 構造体は量子回路における重要な部分であるため、gate 構造体を図 4.4 に示す。

gate 構造体は、各量子ゲートの位置情報、変数情報などがリスト構造形式で格納される。物理的/論理的に何ステップ目に当たるのか、どの変数と関わっているのか、

```
struct gate {
  /* from program definition */
  /*  enum gate_type type; / * #args implicit */
  int type; /* #args implicit */
  struct arg args[MAXARGS]; /* gate operands */
  /* the rest can be generated by various compiler phases */
  int etime; /* execution time slot */
  int logtime; /* logical time */
  struct dependency deps[MAXDEPS];
  int numdeps;
  struct gate *ents[MAXENTS]; /* dependents */
  int numents;
  struct dependency waitingon[MAXDEPS];
  int numwaitingons;
  struct gate *waitedonby[MAXENTS]; /* dependents */
        .
        .
        .
};
```

図 4.4: aquastructs.h 内部の gate 構造体の一部

次に実行する量子ゲートはどれなのかといった情報を格納する。

3. **qasmlex.l**

   flex[16] をベースとした字句解析器である。flex は lex をベースとしており、flex の書式で記述されたコードは字句解析を行うコードを生成する。すなわち、flex の書式で量子回路の字句定義を行うことで、量子回路の字句解析プログラムを生成することが可能である。図 4.5 は qasmlexl.l のコードの一部分である。
   本研究では、第 4.2.2 節で述べる MEASX,MEASY の拡張として利用した。また、拡張子として MBQC アーキテクチャ、Brickwork state アーキテクチャも追加している。新規に量子回路を追加する場合、字句解析だけでなく、構文解析 (次項 qasmparse.y) の拡張も行う必要がある。図 4.5 中最下段の部分は変数 VAR に対する規則を表している。各変数は、アルファベット 1 文字で始まり、2 文字目以降は任意の英数字、およびアンダースコアで構成される合計半角 32 文字以内で構成される。

4. **qasmparse.y**

   Bison[17] をベースとした構文解析器である。Bison は yacc をベースとしており、Bison の書式で記述されたコードは構文解析を行うコードを生成する。qasmparse.y に

```
        .
        .
        .
[0-9]+ { yylval.ival = atoi(yytext); return NUMBER; }

var { return VAR; }
temp { return TEMP; }
nop { return NOP; }
not { return NOT; }
cnot { return CNOT; }
cz { return CZ; }
swap { return SWAP; }
meas { return MEAS; }
measx { return MEASX; }
measy { return MEASY; }
        .
        .
        .
mbqc { return MBQC; }
brickwork { return BRICKWORK; }
init { return INIT; }
zero { return ZEROTOK; }

[a-zA-Z][a-zA-Z0-9_]* { if (debugparser)
            printf("name %s\n",yytext);
        yylval.ival = add_symname(yytext); return NAME; }
        .
        .
        .
```

図 4.5: qasmlex.l のコードの一部分

より、量子回路の構文解析プログラムを生成することが可能である。図 4.6 は qasm-parse.y のコードの一部分である。

MEASX,MEASY は 1 量子ビットの測定を行う量子回路であるため、引数は 1 つである。

```
      .
      .
      .
   | CNOT NAME opt_vloc1 NAME opt_vloc2 opt_loc
     { if (debugparser)
printf("CNOT gate: %d %d\n", $2, $4);
       thisopcode = CNOT;
       argone = $2; argoneloc = argoneloc;
       argtwo = $4; argtwoloc = argtwoloc; }
      .
      .
      .
  | MEASX NAME opt_vloc1 opt_loc { if (debugparser)
   printf("MEASX gate: %d\n", $2);
   thisopcode=MEASX;argone=$2;argoneloc=argoneloc;}
  | MEASY NAME opt_vloc1 opt_loc { if (debugparser)
   printf("MEASY gate: %d\n", $2);
   thisopcode=MEASY;argone=$2;argoneloc=argoneloc;}
      .
      .
      .
```

図 4.6: qasmparse.y のコードの一部分

5. **mbqctarget.c/brickworktarget.c**
   図 fig:quantumcompiler では、中間コードを受け取り、最適化、およびコード生成を行う部分である。これらのファイルの入力には、中間コード生成時、aquastructs.h 内の QuantumProgram 構造体によって定義された形式で格納されているデータを用いる。そして、変数のデータを元にして、量子ビット・量子回路の配置の最適化を行った後に、コード生成を行う。
   本研究では、これらのプログラムの実装を中心に行う。コードの主な流れは、中間コードを読み込み、中間コード内の各ゲートを指定されたアーキテクチャの量子ゲートに対応させて量子回路を出力する形式である。具体的なコードの内容については、第 4.3 節にて示す。

AQUA プログラムの実行例として、AQUA 言語で記述された図 4.7 の CNOT ゲート回路のプログラムを挙げる。

```
1 title "one cnot gate"
2 arch AC
3
4 var A0
5 var B0
6
7 1:  CNOT  A0 B0
8 [EOF]
```

図 4.7: AQUA 言語上で実装した CNOT ゲートのコード

図 4.7 のコードを AQUA プログラム上で実行し、FIG 形式の量子回路に変換したものが図 4.8 である。他のゲートについても同様に出力することができる。

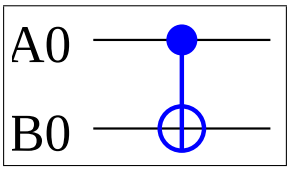


図 4.8: AQUA プログラムによって出力された CNOT ゲートの FIG 画像

### 4.2.2 文法規則

まず、AQUA 言語の文法規則について述べる。まずは、AQUA 言語によって記載されたコードが図 4.9 である。

```
title "3 qubit Teleportation gate"
arch AC

var A0
var B0
var C0

1: H A0
    H B0
    H C0
2: CZ A0 B0
3: CZ B0 C0
4: H A0
    H B0
5: MEAS A0
    MEAS B0
```

図 4.9: AQUA 言語上で実装した 3 量子ビットのテレポーテーションのコード

このコードは、第 2.4 節で述べた 3 量子ビットのテレポーテーション回路を意味する。このコードもまた、AQUA プログラム上で実行可能である。
AQUA 言語上で用いられている基本的な文法を以下の表 4.1 にまとめる。

AQUA プログラム内の構文解析はこの字句・構文規則に従って実行される。
コード内におけるタイトルは、ファイルのタイトル名とは扱いが異なり、コードの内容を簡潔に示すのに用いられる。図 4.9 中のタイトルは、“ 3 qubit Teleportation gate ”である。アーキテクチャとは、量子コンピュータのアーキテクチャのことであり、一番基本となるアーキテクチャは AC 形式である。(要引用) 本研究で扱う量子コンピュータのアーキテクチャは MBQC, BRICKWORK 形式である。図 4.9 は AC アーキテクチャによって記述されている。プログラム内で用いる変数は全て、事前にアーキテクチャ形式に続けて記載する必要がある。図 4.9 中で用いられる変数は A0,B0,C0 の 3 つである。これは、3 量子ビットを使うことを意味する。1, 2, 3 といった数字は、ステップ数のことであり、1 ステップ中でも、異なる量子ビット (変数) であれば複数の回路を実行することが可能である。図 4.9

表 4.1: AQUA 言語文法規則

| 記載 | 内容 | 変数例 |
|---|---|---|
| title | プログラムのタイトル | “ cnot-program ” |
| arch | 量子プログラミングのアーキテクチャ | AC,MBQC など。 |
| var | プログラム内で用いる変数名 | 英文字から始まる任意の半角 32 文字 |
| 1, 2, 3... | 量子プログラム実行上の実時間 | 1 から始まる数値. その後に演算内容が続く。 |
| CNOT | 量子回路をベースとした演算内容 | CNOT の場合、引数を 2 つ持つ |

を例にとると、1 ステップ目は Hadamard ゲートを A0, B0, C0 という名前のそれぞれの量子ビットに適用することを意味する。
現在構文解析することが可能な量子回路の一覧は以下の表 4.2 通りである。

表 4.2: AQUA 言語上で利用可能な量子回路

| 量子回路 | 引数 |
|---|---|
| H | 1 |
| NOT | 1 |
| NOP | 1 |
| CNOT | 2 |
| CZ | 2 |
| CCNOT | 3 |
| SWAP | 2 |
| T | 1 |
| TD | 1 |
| S | 1 |
| MEAS | 1 |
| INIT | 1 |

H は Hadamard ゲートを意味し、CZ は CPahse ゲート、TD は $T^{\dagger}$ ゲートを意味する。これらを AQUA 言語のプログラム内に組み込むことで、変換し、画像として出力できる。これらは、古典演算における演算子に相当する。ここで、各量子回路の取る引数は、第 2.2 節で述べた扱う量子ビットの数と同じである。例えば、H であれば取る引数は 1 つであり、CCNOT であれば取る引数は 3 つである。
また、この回路の組み合わせには、CNOT・Hadamard・Tgate のセット、または Toffoli ゲート (CCNOT) が含まれているため、機能的完全性を満たしている。

以上の内容を用いて、AQUA 言語を AQUA プログラムを用いて実行させる。

### 4.2.3 実装環境

今回、実装環境として aqua-cat サーバを用いて実装した。表 4.3 は本研究における実装環境である。aqua-tools は svn によって管理されており、本研究の拡張の対象とするのはバージョン 21 である。

表 4.3: 実装環境

| | 種別 |
|---|---|
| OS | Ubuntu11.4 |
| gcc | version 4.6.2 |
| aqua-tools | version21 |

## 4.3 AQUA 言語の拡張

まずは、今回実装する AQUA プログラムの拡張の概要について、図 4.10 を用いて説明する。

本研究では、主に橙色の部分の拡張を行う。これは、量子回路の最も基本的なアーキテクチャ形式である AC 形式 (.ac ソース) から、MBQC、Brickwork(.mbqc、.brickwork) のアーキテクチャ形式へのトランスレータである。
.mbqc ソースは複数のソースファイルから構成され、その後の MBQC 実行環境へと応用しやすいようにどの量子ビットがどの関数の役割を持っているのかといった情報も含まれる。これは、mbqctarget.c というファイルを元にして動作する。.brickwork ソースもまた、複数のソースファイルから構成され、およびその先のユニバーサル秘密量子計算に応用しやすいように、測定角度の情報、およびどの関数がどの役割を持っているのかといった情報が含まれる。これは、brickworktarget.c というファイルを元にして動作する。

.mbqc 形式、.brickwork 形式は以下に示す順番の内容で動作する。

- output ファイル定義
  出力するファイル名の定義、およびファイルを書き込み専用で開く。

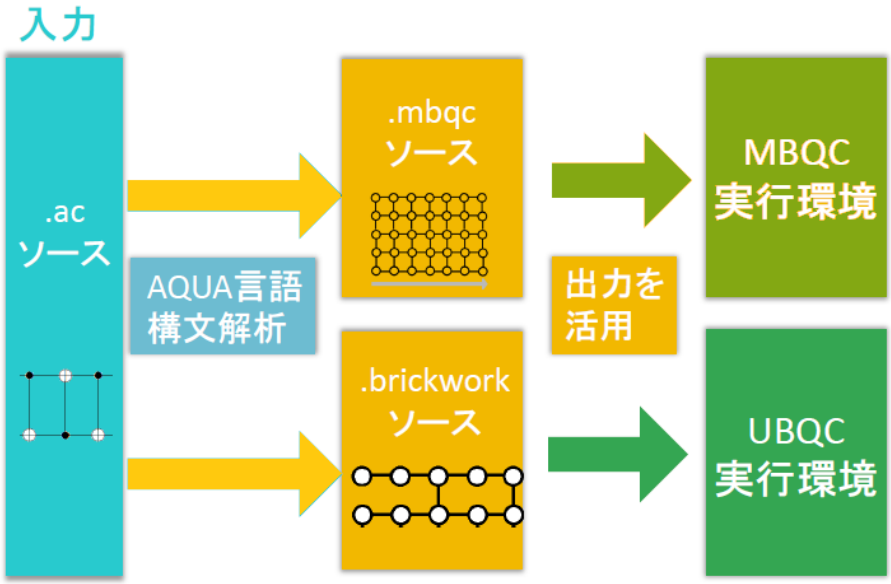


図 4.10: 本研究において実装するプログラムの処理概要

- ysize の決定
  本研究のアーキテクチャ形式であれば、ysize は MBQC、Brickwork state において元の入力回路の変数が定まれば一意に定まる。なお、xsize は横方向にどれだけの量子ビットが必要か、ysize は縦方向にどれだけの量子ビットが必要かを示し、xsize×ysize の値をプログラムにおける合計の量子ビットと定める。つまり、本研究で生成される量子回路は必ず長方形になる。

- 中間コードに含まれる gate リスト構造体を解析
  中間コードにリスト構造で格納されている gate 構造体のデータを順に処理する。gate 構造体にはステップ数の情報も含まれているため、ステップ数毎に仕切り板をつける。仕切り板とは、ある 1 ステップ中にどれだけの横方向に量子ビットが必要なのかといった情報である。仕切り板は、各ステップにおける一番サイズを必要とする量子ゲートに合わせなくてはならない。
  gate 構造体の内部は、その他にも量子ゲートの識別後に対象とする量子ビットの数に応じて以下のアルゴリズムで解析を行う。全ての量子ゲートを解析後、合計の量子ビット数 (sumqubits) を計算する。

  - 1 量子ビットのゲートの場合
    Hadamard ゲートを代表とする 1 量子ビットのゲートの場合、仕切り板のサイズ判別を行い、指定された番地にどの回路が入るか決定し、専用の配列に格納する。現バージョンで配列を用いているため、配置可能な量子ビットの数に制約がある。
  - 2 量子ビット以上のゲートの場合

CNOT ゲートを代表とする 2 量子ビットのゲートの場合、まずどちらの量子ビットが Control ビットであるか判別を行う。その後、Target ビットが隣接している場合は仕切り板のサイズ判別を行い、指定された番地にどの回路が入るか決定する。しかし、隣接していない場合はその分 SWAP ゲートを用いた等価回路に変換した後に処理する必要がある。本研究で用いた非隣接 CNOT ゲートの SWAP ゲートを用いた等価回路への変換方法については後述する。

- コード整形
  本研究では対象としていないが、複数ゲート間の最適化を必要とする場合、この段階で行う。
  その後、何も量子ゲートが割り振られていない部分に I ゲートと呼ばれる何も操作を行わないゲートを配置する。最後に、量子ビット列を ysize 分 1 列加えることで完成する。この 1 列は EOQ 列と呼び、MBQC、Briwkwork state 回路における出力された量子ビット列を意味する。

- コード生成
  中間コードを解析した結果、生成された配列のデータを元にして、AC 形式で量子回路を生成する。まず、変数の定義であるが、左方向から一番上から下へと順に M1、M2、M3... と割り振られていく。変数番号の割り振り例を図 4.11 に示す。
  そして、割り振られた変数に合わせて各量子ビットに存在する量子ゲートの情報を

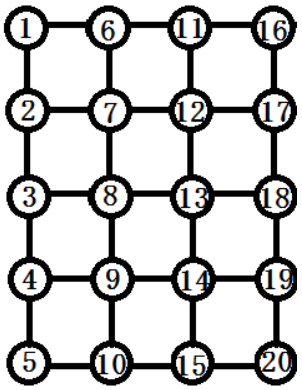


図 4.11: MBQC における量子ビットの番号割り振り

元に量子回路を生成する。

以上が mbqctarget.c、brickworktaarget.c のプログラムの概要である。各アーキテクチャに特化した実装内容の部分については後述する。

## 4.4 AQUA 言語の拡張：MBQC 形式

### 4.4.1 概要

AQUA 言語の拡張として、MBQC アーキテクチャ方式の量子回路を生成、および量子回路を MBQC 形式へと変換するするプログラムを実装した。プログラムの処理内容を図 4.12 に示す。AQUA プログラムの MBQC アーキテクチャへの拡張として、mbqctarget プ

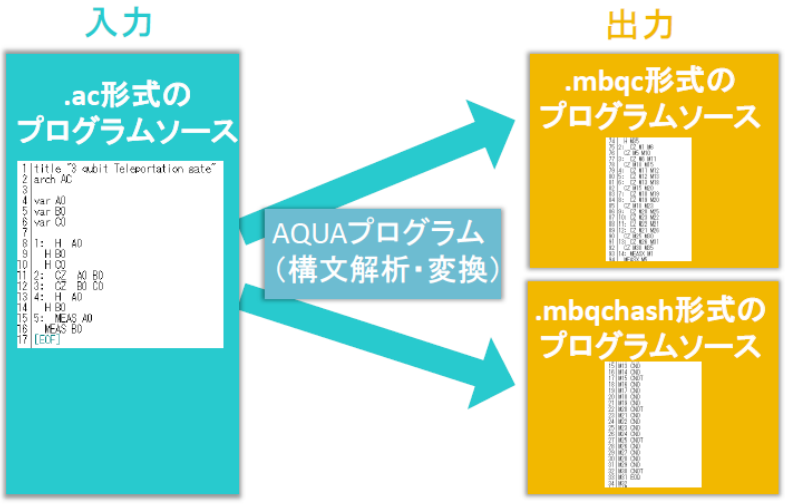


図 4.12: MBQC 変換コンパイラの処理内容

ログラムを追加した。 mbqctarget プログラムは、主に.mbqc ファイルと.mbqchash ファイルを生成する。.mbqc ファイルは MBQC アーキテクチャ形式で記述された量子回路であり、.mbqchash ファイルは各変数がどのゲートの役割を果たすかについて記述されたファイルである。
今回、MBQC 変換プログラムにて実装した回路は以下の通りである。これらの量子回路図については、付録にて記す。

- H
- CNOT
- SWAP

なお、変数が論理量子ビットにおいて隣接していない非隣接状態の CNOT ゲートについても実装を行った。(第 4.4.2 節)
AQUA 言語への MBQC 形式の拡張に加えて、本研究では MBQC の量子回路を実現するために、以下の 2 つの回路を使えるように追加した。

- MEASX
- MEASY

これらのゲートは、それぞれ X 軸方向での測定、Y 軸方向での測定を意味する。MBQC での主な測定方法は X 軸方向の測定、Y 軸方向での測定、そして adaptive qubit の 3 種類である。これら 2 つの回路は、MBQC 上で測定として汎用される回路であり、重要な役割を果たす。
MEASX,MEASY の AQUA プログラムによる FIG 形式の出力は以下の通りとなる。（図 4.13、図 4.14）
付録 A に示す MBQC の量子回路にも、頻繁に MEASX、MEASY は用いられている。

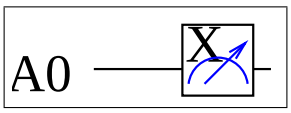


図 4.13: MEASX の出力

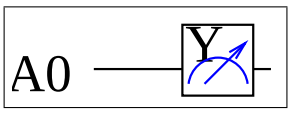


図 4.14: MEASY の出力

このプログラムを使い、図 4.7 に示される CNOT ゲートのコードを変換した場合の.mbqc 形式のファイルは以下のソースコード 4.15,4.164.17 である。なお、このコードは、14 ステップ、および 35 変数から構成される。
まず、ソースコード 4.15 を掲載する。

```
title "one cnot"
arch AC

var M1
var M2
var M3
       .
       .
       .
var M33
var M34
var M35
```

図 4.15: MBQC 上で動作する CNOT ゲートの回路（1）

プログラムのタイトルは、元の入力ソースのタイトルと同一のタイトルである。そして、このプログラムには変数が 35 個ある。これは、このプログラムが 5 行 7 列の合計 35 量子ビットから構成されているためである。
次に、図 4.16 を掲載する。

```
1: H M1
    H M2
    H M3
       .
       .
       .
    H M33
    H M34
    H M35
2: CZ M1 M6
    CZ M5 M10
3: CZ M6 M11
    CZ M10 M15
4: CZ M11 M12
5: CZ M12 M13
6: CZ M13 M18
       .
       .
       .
10: CZ M23 M22
11: CZ M22 M21
12: CZ M21 M26
    CZ M25 M30
13: CZ M26 M31
    CZ M30 M35
```

図 4.16: MBQC 上で動作する CNOT ゲートの回路（2）

1 ステップ目の H ゲートは、MBQC における初期処理である。2 ステップ目から 13 ステップ目までの CZ ゲートは、量子ビット間の相互作用の役割を持つ。例えば、2 ステップ目最初の“ CZ M1 M6 ”は、量子ビット M1 から量子ビット M6 への量子情報の移動の役割を担う。
CZ ゲートを正しく生成するために、mbqctarget 内の中間コードを元にして生成された配

列の情報 (.mbqchash ファイルとほぼ同一) を元にしている。配列には、各番地にどのゲートが割り振られているか示されているため、1 列ごとに処理を行う過程で各行にゲートカウンタを割り振り、ゲートカウンタが進むと次の列に対応する CZ ゲートの配置に進む。例として、図 4.16 の場合、量子ビット M1 から量子ビット M30 までが CNOT ゲートの番地であり、そのサイズは 5×6 であるため、ゲートカウンタの値は 0 から 5 までの 6 段階である。ゲートカウンタの 6 段階目では、CZ ゲートの処理を終えた後に、図 4.17 に示される測定が行われる。

```
14: MEASX M1
    MEASX M5
    MEASY M6
    MEASX M10
    MEASY M11
    MEASY M12
    MEASX M13
    MEASX M15
    MEASY M18
    MEASY M19
    MEASX M20
    MEASY M21
    MEASY M22
    MEASX M23
    MEASX M25
    MEASY M26
    MEASX M30
```

図 4.17: MBQC 上で動作する CNOT ゲートの回路（3）

測定は、ゲートカウンタ 6 段階目で 1 ゲートの分を一括で行う方式である。ゲートに対応する測定の組み合わせを出力する。この例の場合、図 4.18 に示すモデルを採用した。[10] 図中のピンク色の丸は MEASX で測定することを意味し、赤い丸は MEASY で測定することを意味する。そして、橙色の線はこの回路中でどの量子ビットが相互作用を及ぼしているかを表す。

次に.mbqchash ファイルの内容を図 4.19 に示す。
.mbqchash は、どの変数がどの量子ゲートに相当するか記述する。この内容は、図 4.18 の CNOT ゲートの量子回路を元にしている。まず、$TIME06$ は 1 仕切りにおける xsize を意味する。今回の場合、0 番目のゲート列処理は 6 列の量子ビットから構成されることがわ

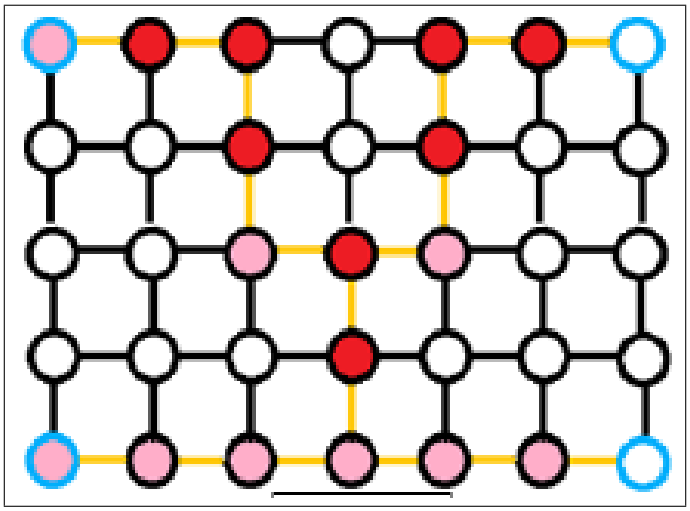

図 4.18: MBQC アーキテクチャで実装された CNOT ゲート

かる。それに続いて、量子ビット M1 を初めとし、各量子ビットにどのゲートが割り振られるかが示される。
図中の“ CNO ”は Control ビットを指し、“ CNOT ”は Target ビットを指す。この内容は、中間コードを元にして生成された配列の情報と一致する。なお、プログラム中では本研究で用いる MBQC アーキテクチャの列の大きさが $4n \times 1$ に固定される性質を利用しているため、AC 形式のコード出力中は M1、M5、M6、M10、M11、M15... と、1 列につき 4 つ飛ばしで読み込んでいく。
そして、M31 から M35 は EOQ(End of Qubit)、すなわちこの量子回路における終端列であることを意味する。終端列は必ず 1 列であり、この列の量子測定は他の列の量子ビットの量子測定終わった後になる。そして、最後にプログラム中で使用した合計の量子ビット数 QSUM の数が記述されている。今回の場合、$(6+1) \times 5 = 35$ 量子ビットである。
なお、本研究にて実装を行った回路の図、回路情報は付録にて記す。

```
TIME0 6
M0
M1 CNO
M2 CNO
M3 CNO
M4 CNO
M5 CNOT
M6 CNO
       .
       .
       .
M27 CNO
M28 CNO
M29 CNO
M30 CNOT
M31 EOQ
M32
M33
M34
M35 EOQ
QSUM 35
```

図 4.19: MBQC 上で動作する CNOT ゲートの回路（.mbqchash ファイル）

### 4.4.2 MBQC 形式における非隣接 CNOT ゲートの実装

非隣接 CNOT ゲートの量子回路のうち、3 neighbor の CNOT ゲートのモデルは既に存在する。[10] この量子回路を参考に、MBQC 上で N neighbor の CNOT ゲートのモデルを実装した。図 4.20、4.21 は、本研究で実装した 3 neighbor CNOT ゲートの MBQC モデルである。このゲートに用いられている量子ビットは、$9 \times 7 = 63$ 量子ビットである。

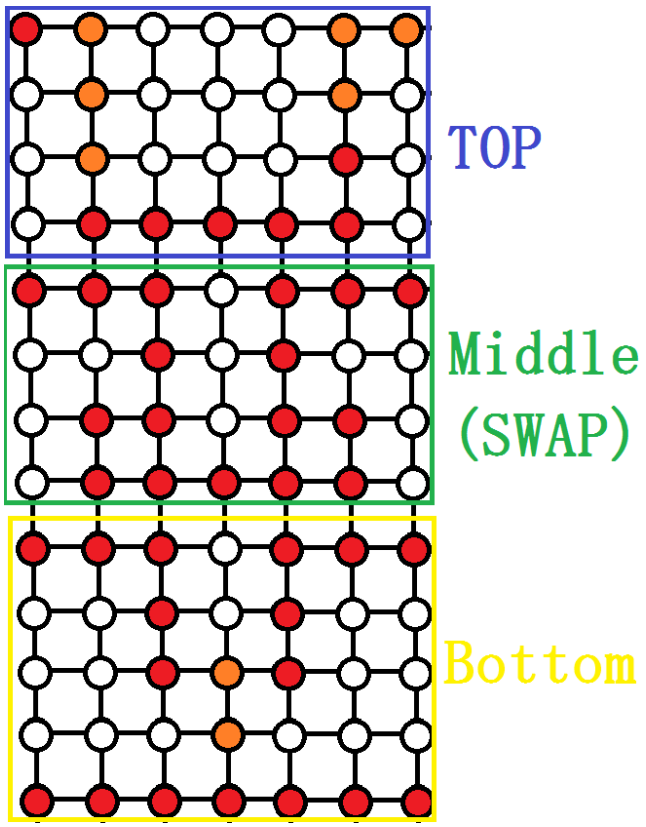


図 4.20: MBQC アーキテクチャで実装した 3 neighbor の CNOT ゲート

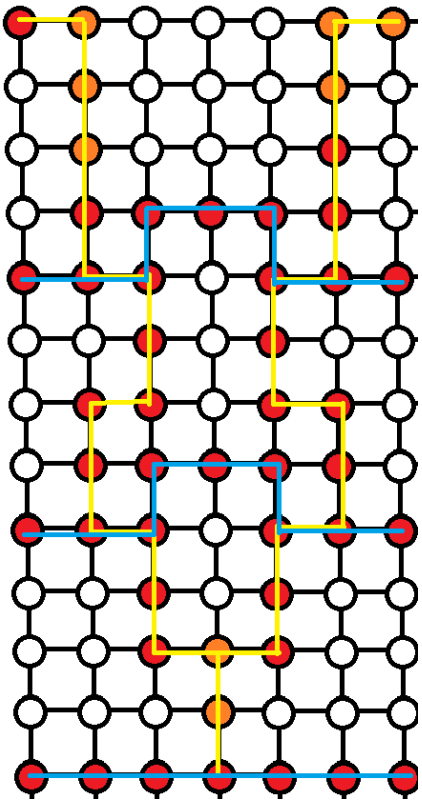

図 4.21: MBQC アーキテクチャで実装した 3 neighbor の CNOT ゲート
量子ビット間のエンタングルメントが行われる部分

このモデルは、N neighbor の場合に対応するために、３つのパーツに分けることが可能である。この３つの部分を組み合わせることで N neighbor CNOT ゲートを構成できる。図 4.21 の黄色の線は Target ビットに向かう相互作用であり、青色の線は SWAP される量子ビットの相互作用である。この図 4.20 では赤色が MEASX、橙色が MEASY の測定を意味する。
この TOP パーツ、Middle パーツ、Bottom パーツを組み合わせることで、図 4.22 の N neighbor の CNOT ゲートが実現する。図中右側の数値は、各パーツの ysize を意味する。TOP パーツ、Middle パーツがが $4 \times 7 = 28$ 量子ビット、そして Bottom パーツが $5 \times 7 = 35$ 量

子ビットで構成される。

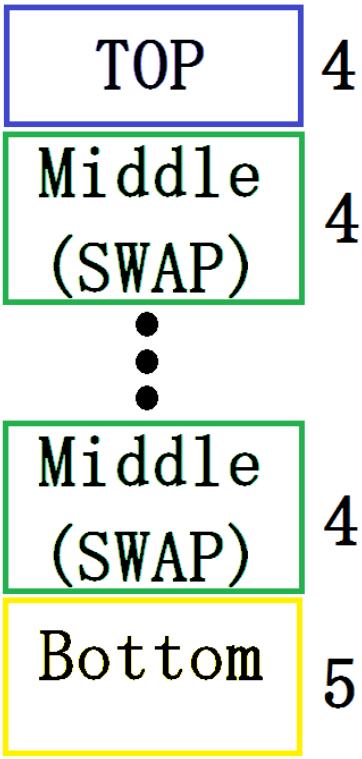


図 4.22: MBQC アーキテクチャで実装された N neighbor の CNOT ゲート

1 つ目の“ TOP ”のパーツは、Control ビットに相当する部分である。この TOP パーツは Middle のパーツ、もしくは Bottom のパーツと連結する。Bottom パーツと連結するのは 2 neighbor CNOT ゲートの場合であり、Middle パーツを 1 度も使用しない。また、このパーツは 1 回のみ使用される。
Middle パーツは SWAP ゲートの役割を果たす。Control から Target へと情報を伝搬する役割を果たす。そのため、Middle パーツは非隣接数が大きいほど利用する回数が増える。N neighbor の CNOT ゲートであれば、Middle パーツは N-2 回使用する。
この Bottom パーツは、TOP パーツと同様、1 回のみ使用する。最下段に配置され、Target ビットに相当する役割を持つ。
表 4.4 に、各パーツに用いられる量子ビット、回路の情報を記す。ここでの実量子ビットとは、CZ ゲートによる相互作用が行われる量子ビットのことであり、その後に量子測定が行われる量子ビットの合計でもある。

AQUA プログラムによって出力される非隣接 CNOT ゲート (2、3、4 neighbor) の量子回路は付録 A に掲載する。

表 4.4: 非隣接 CNOT ゲートを構成するパーツの情報

| 内容 | TOP パーツ | Middle パーツ | Bottom パーツ |
|---|---|---|---|
| ysize | 4 | 4 | 5 |
| xsize | 7 | 7 | 7 |
| 実量子ビット | 13 | 17 | 19 |
| MEASX の数 | 7 | 17 | 17 |
| MEASY の数 | 6 | 0 | 2 |
| CZ ゲートの数 | 13 | 20 | 24 |

## 4.5　AQUA 言語の拡張：Brickwork 形式

### 4.5.1　概要

AQUA 言語の拡張として、Brickwork state に対応した量子回路の生成を行えるようにした。図 4.23 は、Brickwork state のプログラムの概要である。

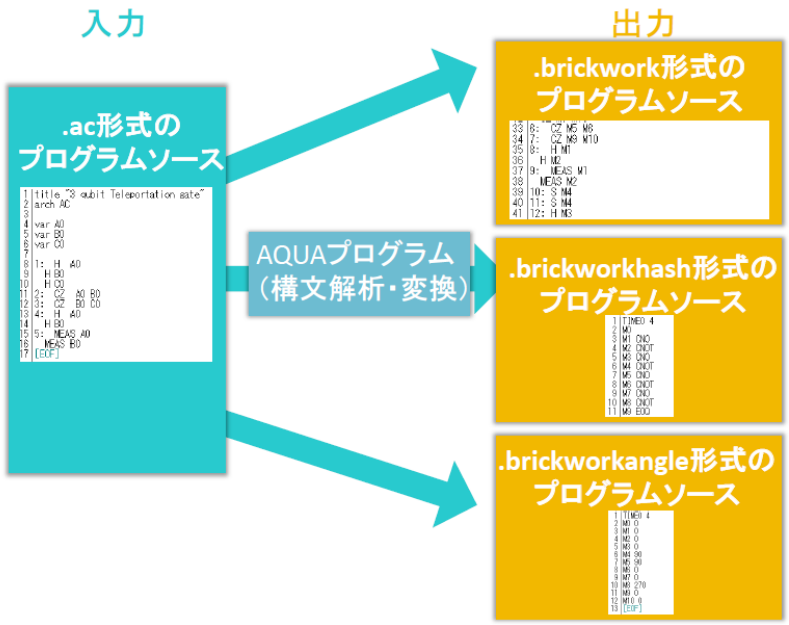


図 4.23: Brickwork 変換プログラムの処理内容

AQUA プログラムの Brickwrok state アーキテクチャへの拡張として、brickworktarget プログラムを追加した。brickworktarget プログラムは、主に.brickwork ファイル、.brickworkhash ファイル、.brickworkangle ファイルを生成する際に用いる。.brickwork ファイルは Brickwork アーキテクチャ形式で記述された量子回路である。.brickworkhash ファイルは各変数がどのゲートの役割を果たすかについて記述されたファイルである。そして、.brickworkangle ファイルは、各量子ビットがどの角度で測定を行われるかを示す情報が含まれており、実際のユニバーサル秘密量子計算では関数に相当する。各測定角度に暗号化を施し、サーバに送信することで、秘密計算を実行できる。秘密計算において、暗号する角度は $\pi/4$ 単位と定められている。つまり、測定に用いられる角度は $\frac{\pi}{4}$、$\frac{\pi}{2}$、$\frac{3\pi}{4}$、$\pi$、$\frac{5\pi}{4}$、$\frac{3\pi}{2}$、$\frac{7\pi}{4}$、$2\pi = 0$、の 8 種類である。これらの各測定を実現するために、T ゲートを用いる。T ゲートは $\frac{\pi}{4}$ 単位で Z 軸方向の回転を行うことができるため、T ゲート演算を任意の回数繰り返すことで 8 種類の回転角度を表現することができる。

今回、Brickwork 変換プログラムにて実装した回路は以下の通りである。これらの量子回路図、詳細については付録にて記す。CNOT ゲートについては、MBQC 形式同様に非

隣接ゲートにも対応した。なお、Brickwork における非隣接 CNOT ゲートは 3 種類のバージョンが利用可能である。これらのバージョンの違いは、Brickwork 上で SWAP ゲートをどのようにして配置するかの違いである。第 4.29 にその詳細を述べる。

- H
- CNOT
- SWAP
- T
- TD
- S
- CCNOT

図 4.24、図 4.25 は brickworktarget によって出力された.brickwork 形式のコードである。図 4.24 の部分は.mbqc 形式と同一である。しかし、図 4.25 からは MBQC とは異なり、一定の方法に従って相互作用、測定が行われる。なぜなら、Brickwork state は xsize と ysize が決定することでその上で実行される内容に関わらず形が一定であるためである。今回の場合、xsize が 5、ysize が 2 であるため、1 パーツ分の Brickwork state に相当する。なお、量子ビットへの番号割り振りは MBQC で用いた図 4.11 の方法と同一である。
具体的には、まず、ステップ 2 からステップ 7 までで各量子ビット間の相互作用を行う。そして、測定時に Z 軸方向で測定を行うための T ゲートによる任意の回転処理、Hadamard ゲートによるステップ 1 で行った重ね合わせの状態の復元を経ることで Z 軸方向の測定を行う。MEAS が Z 軸方向の測定を意味する。これらの処理は、列単位で行われ、一番最初の量子ビット列を除いて 4 の倍数で割り切れる場合に、測定処理を 4 列分まとめて行う。なお、ここで 1 パーツ内で一度に測定を行っていない理由として、MBQC 全般における、測定結果を元にした修正処理が挙げられる。前列の Z 軸方向の測定によって得られた結果に応じて、後列の量子ビット群に指定のオペレータをかけることで修正を行う。ここでのオペレータとは、X ゲートや Z ゲートのことである。量子ゲートによる修正処理は今回の量子回路の内容においては反映されていないが、ステップを分けることによって、指定のオペレータによって修正を行える時間を設けられるようにした。

```
title "one cnot"
arch AC

var M1
var M2
var M3
var M4
var M5
var M6
var M7
var M8
var M9
var M10

1: H M1
    H M2
    H M3
    H M4
    H M5
    H M6
    H M7
    H M8
    H M9
    H M10
```

図 4.24: Brickwork state 上で動作する CNOT ゲートの回路 (.brickwork ファイル)(1)

```
2: CZ M1 M3
    CZ M2 M4
3: CZ M3 M5
    CZ M4 M6
4: CZ M5 M7
    CZ M6 M8
5: CZ M7 M9
    CZ M8 M10
6: CZ M5 M6
7: CZ M9 M10
8: H M1
    H M2
9: MEAS M1
    MEAS M2
10: T M4
11: T M4
12: H M3
    H M4
13: MEAS M3
    MEAS M4
14: T M5
15: T M5
16: H M5
    H M6
17: MEAS M5
    MEAS M6
18: T M8
19: T M8
20: T M8
21: T M8
22: T M8
23: T M8
24: H M7
    H M8
25: MEAS M7
    MEAS M8
```

図 4.25: Brickwork state 上で動作する CNOT ゲートの回路 (.brickwork ファイル)(2)

次に、.brickworkhash、.brickworkangle の内容について図 4.26、4.26 を用いてそれぞれ示す。.brickworkhash の内容は、第 4.4 節で述べた.mbqchash の内容と書式が同じである。一方、.brickworkangle は Brickwork state 特有の内容である。

```
TIME0 4
M0
M1 CNO
M2 CNOT
M3 CNO
M4 CNOT
M5 CNO
M6 CNOT
M7 CNO
M8 CNOT
M9 EOQ
M10 EOQ
EOF 10
```

図 4.26: Brickwork state 上で動作する CNOT ゲートの回路 (.brickworkhash ファイル)

```
TIME0 4
M0 0
M1 0
M2 0
M3 0
M4 90
M5 90
M6 0
M7 0
M8 270
M9 0
M10 0
```

図 4.27: Brickwork state 上で動作する CNOT ゲートの回路 (.brickworkangle ファイル)

.brickworkangle は各量子ビットを測定する際に、どれだけの回転を必要とするのかといった情報を主に格納する。角度は 0 を始めとして 45 度刻みで表現され、360 度で 1 回転、0 度に戻る。このコードを例にとると、量子ビット M4・M5 を $\frac{\pi}{2}$ 回転させた後に測定し、M8 を $\frac{3\pi}{2}$ 回転させた後に測定する。その他の量子ビットは何もせずに測定を行う。なお、このデータは暗号化が施される前のデータであるため、秘密計算を行う場合、暗号化を施してからサーバに情報を送信しなくてはならない。
なお、上記のコードは Brickwork state における CNOT ゲート [12] に相当し、量子ビットを使った表現の場合、図 4.28 の通りになる。

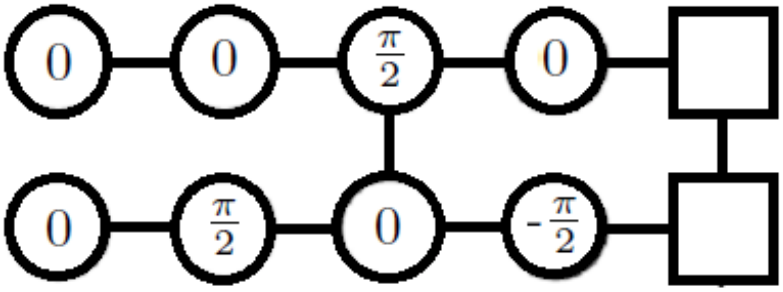


図 4.28: Brickwork state における CNOT ゲート

実装にあたって、大規模回路に対応できるように、自動的に 2 量子ビット以上の相互作用が Brickwork のパーツの部分に割り振られるように工夫した。これは、2 量子ビット以上を対象とする量子ゲートにおいて変数情報を解析し、量子ゲートが割り振れない位置に存在する場合は後のゲートを $4 \times ysize$ 分、すなわち 1 パーツ分の I ゲートを挿入して対応する。また、最終 TIMES 以外の TIMES の値を読み込み、それらが 8 で割り切れない場合 (各仕切りにおいて 1 パーツ分しか割り振られていない場合) もまた、1 パーツ分の I ゲートを挿入して対応する。この 2 つの規則によって、2 量子ビット以上の相互作用が確実に Brickwork のパーツに割り振られる。

### 4.5.2 Brickwork state における SWAP ゲート回路

Brickwork state において非隣接 CNOT ゲートの実装をするにあたって、まずは今までに実装されていなかった SWAP ゲートの実装を行った。図 4.29 は Brickwork state によって実装された $xsize = 20$ の SWAP ゲートである。ここでの $xsize$ は SWAP ゲート本体の大きさであり、以降も同様に、最後に付加される 1 列の出力用量子ビットを除外して考える。

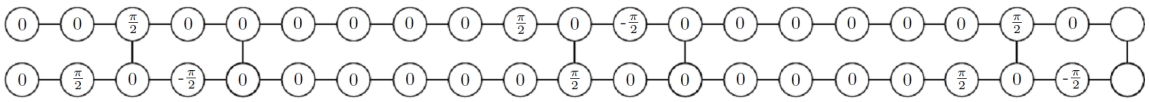

図 4.29: サイズ 20 の SWAP ゲート : Brickwork state

これは、第 2.2.7 節で述べた SWAP ゲートの等価回路である CNOT ゲートを 3 つ組み合わせた回路である。Brickwork state の特性により、このゲートは以下の図 4.30 の形でマッピングされる。

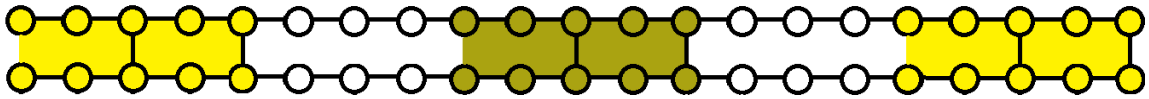

図 4.30: サイズ 20 の SWAP ゲートの構造 ： Brickwork state

黄色の部分が通常の CNOT ゲートであり、黄褐色の部分が CNOT ゲートにおいて、Control ビットと Target ビットを反転させたものである。この SWAP ゲートは、$20 \times 2 = 40$ 量子ビットから構成され、既存の $4 \times 4 = 16$ の MBQC における SWAP ゲート (図 3.4) と比較すると、かなり多くの量子ビットを要する。
そこで、Brickwork state における SWAP ゲートをより使いやすくするものにするために、Brickwork state における依存関係のない場での結合性 [18] を活用する。これは、同一の量子ビット (行) を扱う量子ゲート間で成り立ち、それらの測定角度を足し合わせることができる性質である。例として、Hadamard ゲートと CNOT ゲートの結合を図 4.31 に示す。また、元となる量子回路を図 4.32 に示す。

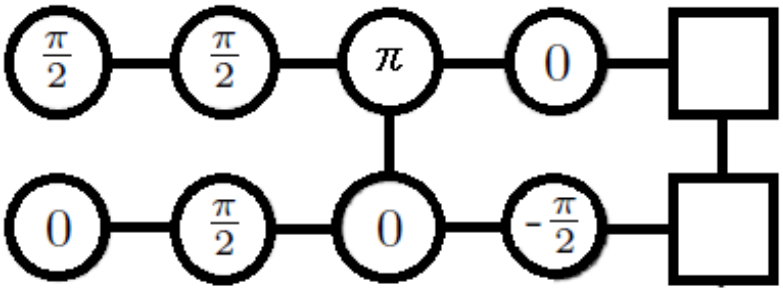


図 4.31: Brickwork state における Hadamrd ゲートと CNOT ゲートの結合

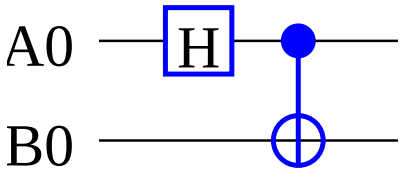


図 4.32: Brickwork state における結合の元となる回路

1、3、5 番目の量子ビットが Hadamard ゲートの量子測定角度に、4、5、8 番目の量子ビットが CNOT ゲートの量子測定角度に相当する。ここで、5 番目の量子ビットは Hadamard ゲート、CNOT ゲートの双方の影響を受けているため、その角度は $\frac{\pi}{2} + \frac{\pi}{2} = \pi$ となる。これは、Brickwork state における CCNOT ゲートの実装 [18] の際にも用いる手法である。この法則を利用して、図 4.29 の $xsize = 20$ の SWAP ゲートを図 4.33 の $xsize = 4$ の SWAP ゲートに短縮した。

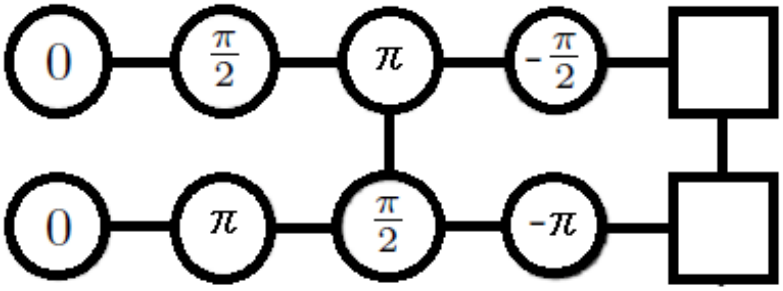


図 4.33: Brickwork state におけるサイズ 4 の SWAP ゲート

3、6、7 番目の量子ビットは 2 番目の CNOT ゲートの影響を受けており、4、5、8 番目の量子ビットは 1 番目と 3 番目の CNOT ゲートの影響を受ける。表 4.5 に、既存の MBQC における SWAP ゲート、サイズ 4、20 で実装された Brickwork state における SWAP ゲー

トをまとめる。また、以降の内容は全てサイズ 4 の SWAP ゲートを元にした実装・評価を行う。

表 4.5: MBQC、Brickwork state における SWAP ゲートの比較

| 内容 | MBQC | Brickwork：サイズ 4 | Brickwork：サイズ 20 |
|---|---|---|---|
| ysize | 5 | 2 | 2 |
| xsize | 4 | 4 | 20 |
| 実量子ビット | 14 | 8 | 40 |
| CZ ゲートの数 | 17 | 10 | 50 |
| ステップ数 | 13 | 29 | 101 |

### 4.5.3 Brickwork state における非隣接 CNOT ゲート

第 4.5.3 節で述べた SWAP ゲートを活用し、Brickwork state における非隣接 CNOT ゲートの実装を行った。ここでは、brickworktarget.c において実装したい SWAP ゲートの配置方法について述べる。
まず、一番最初のバージョン 1 は V 字型に SWAP ゲートを配置するものである。図 4.34 に 2 neighbor の CNOT ゲートを、図 4.35 に 3 neighbor の CNOT ゲートを示す。

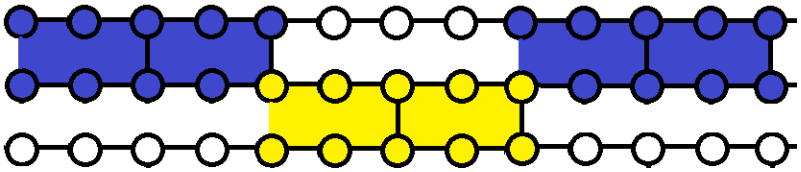

図 4.34: 2 neighbor CNOT ゲート (ver1):Brickwork state

図中の青色の長方形で塗りつぶされた部分には SWAP ゲートを、黄色の長方形で塗りつぶされた部分には CNOT ゲートを配置されることを意味する。以降の非隣接 CNOT ゲートにおいても同様の表記を用いる。この 2 neighbor の CNOT ゲートの形は以降のバージョン 2、バージョン 3 においても用いる形である。

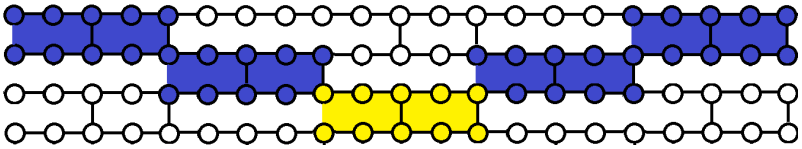

図 4.35: 3 neighbor CNOT ゲート (ver1):Brickwork state

バージョン 1 は SWAP ゲートを V 字型に配置し、CNOT ゲートの操作を最下段にて行う。図 2.11 で示した SWAP ゲートの配置方法と同一の配置方法を用いている。今回、量子ビットの使用サイズは長方形で見積もられるため、この配置方法は空白部分が大きくなってしまい、一見すると非効率な方法に見える。しかし、実際に大規模な量子回路を組む場合、他の複数の量子ゲートと組み合わせて作成するため、他の量子ゲートを空白部分に埋めることで最適化が可能である。つまり、このバージョン 1 の配置方法は多くの複数の量子ゲートを挿入する場合に応用が利く。

次に、バージョン 2 の SWAP ゲートの配置方法について述べる。バージョン 2 は X 字型に SWAP ゲートを配置する方法であり、Brickwork state の交互に 2 量子ビットの操作が行えるパーツが現れる特性を活かした方法である。図 4.36 に 3 neighbor の CNOT ゲートを、

図 4.37 に 4 neighbor の CNOT ゲートを、図 4.38 に 5 neighbor の CNOT ゲートを示す。

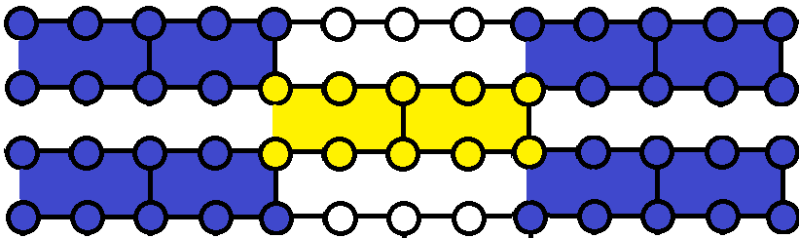

図 4.36: 3 neighbor CNOT ゲート (ver2):Brickwork state

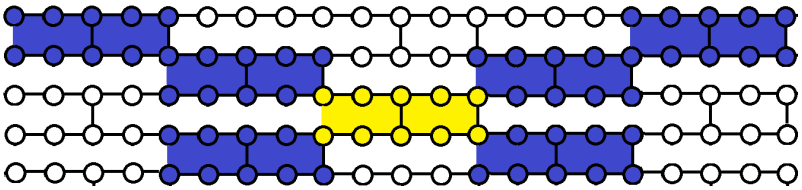

図 4.37: 4 neighbor CNOT ゲート (ver2):Brickwork state

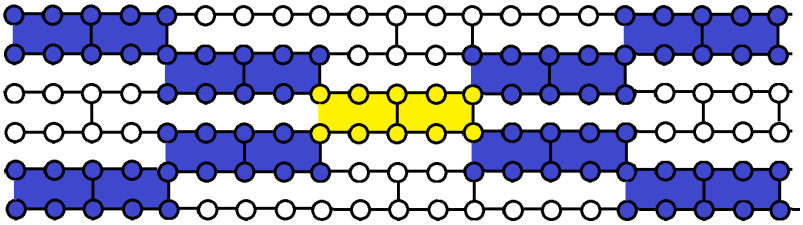

図 4.38: 5 neighbor CNOT ゲート (ver2):Brickwork state

非隣接数が奇数の場合は、上下共に同じ数の SWAP ゲートを置く点対称な形になる。非隣接数が偶数の場合、非隣接数が 1 つ大きい奇数の配置から、最下段の SWAP ゲートを 1 組省くことで配置を実装した。この配置手法は、少ない非隣接度合の元では SWAP ゲートを敷き詰めることができるが、非隣接度合が大きくなるほど空白部分が大きくなる。その場合、空白部分に他の量子ゲートを挿入することで最適化を行うことが可能である。

そして、バージョン 3 は非隣接度合が大きい場合においても、SWAP ゲートを隙間なく詰め込まれる配置方法である。図 4.39、4.40 は 5 neighbor の CNOT ゲートである。

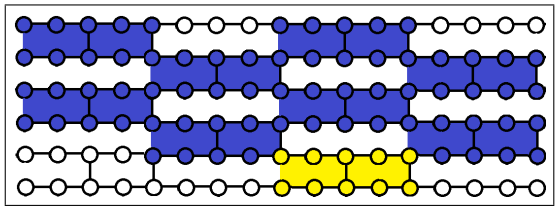

図 4.39: 5 neighbor CNOT ゲート (ver3):Brickwork state

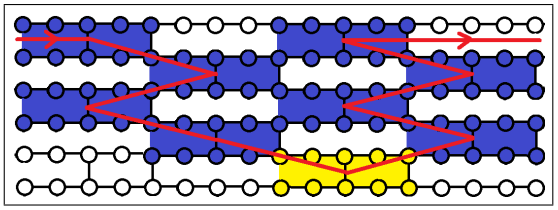

図 4.40: 5 neighbor CNOT ゲート (ver3) の挙動:Brickwork state

Brickwork state の特徴により、SWAP ゲートは交互に配置される。n-1 列に SWAP ゲートを敷き詰め、それらは図 4.40 に示される通りに連結し、非隣接の CNOT ゲートを実現する。非隣接数が奇数の場合と偶数の場合で配置方法、連結方法が異なってくる。奇数の場合は図 4.39、4.40 の通りであり、偶数の場合、図 4.41、4.42 によって示される。

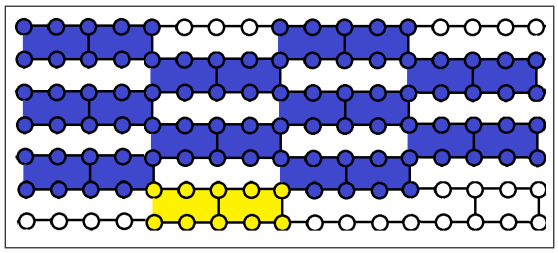

図 4.41: 6 neighbor CNOT ゲート (ver3):Brickwork state

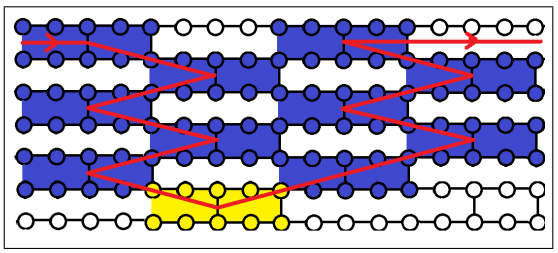

図 4.42: 6 neighbor CNOT ゲート (ver3) の挙動:Brickwork state

非隣接数が奇数の場合との最大の違いは、CNOT ゲートの位置である。奇数の場合、3 ブロック列目に最下段に CNOT ゲートが配置されるが、偶数の場合は 2 ブロック列目最下段に配置される。

以上の 3 バージョンの SWAP ゲートの配置方法を実装した。この SWAP ゲートのバージョンは、brickworktarget.c を含む aqua-tools をコンパイルする時に brickworktarget.c 内部にて選択する必要がある。

# 第5章　評価

本章では AQUA 言語をの拡張として構築したシステムについての評価をまとめる。

## 5.1　AQUA 言語の拡張に対する動作検証

AQUA 言語の拡張として実装した mbqctarget、および brickworktarget において、各量子ゲートおよび複数の量子ゲートが集まった量子回路がそれぞれ正しく動作するかの定性評価を行った。本節では、その評価内容についてまとめる。

### 5.1.1　実装した量子ゲートの定性評価

各量子回路を AQUA プログラムによって実行し、それぞれが正しく出力されるか調査した。これは、単一の量子ゲートが正しく出力されるかを調べる定性評価である。付録 A に、出力された量子ゲートの各図を示す。それらの量子ゲートを以下の方法によって正しく生成されるか判断した。

- **評価の対象とする量子ゲート**
  MBQC の場合、Hadamard ゲート、CNOT ゲート、SWAP ゲート、2,3,4 neighbor の CNOT ゲートの 6 種類である。4 neighbor 以降は、それ以前の回路を応用して生成できるため、4 neighbor までを対象とした。
  Brickwork state の場合、Hadamard ゲート、CNOT ゲート、SWAP ゲート、T ゲート、S ゲート、2,3,4 neighbor の CNOT ゲート、CCNOT ゲートの 9 種類である。非隣接 CNOT ゲートについては、バージョン 1、バージョン 2、バージョン 3 の 3 種類を 2,3,4 neighbor の場合について確認した。これらの詳細については後述する。

- **CZ ゲートの配置**
  MBQC、Brickwork state アーキテクチャにおいて、CZ ゲートは量子ビットの相互作用を意味する。出力された図において、CZ ゲートが正しく連結されているか確認した。連結されているかの確認は、最左列の量子ビットから最右列の量子ビット列まで到達できるかによって判定する。

- **測定の種類・およびその位置**
  MBQC であれば、MEASX、MEASY が、Brickwork state であれば T ゲートがそれらに該当する。この双方の量子ゲートが正しい位置に配置されているかどうか確認した。また、MEAS 系列の量子ゲートの後に続くステップで量子ゲートが実行されていないかどうかといった順序の整合性を確認した。

- **mbqchash、brickworkhash ファイル**
  対象とする量子ゲートについて、対応する mbqchash、brickworkhash のデータ、およびその量子回路に用いられる量子ビット数が正しく出力されるか確認した。brickworkstate については、更に brickworkangle についての確認も行った。特に注意深く確認したのは、SWAP ゲートの配置である。デバッグの効率化を図るため、mbqctarget、brickworktarget にはそれぞれ mbqclog、

### 5.1.2 大規模な回路による実験

量子回路は、通常単体として存在するものでなく、複数のゲートを組み合わせることで成立する。そこで、第 2.1.3 節の図 2.1 で示した回路を動作させることによる定性評価を行った。
本研究では、MBQC における CCNOT ゲートの実装 [10] を対象外とするため、MBQC の場合、既に考案された量子ゲート [10] における量子ビット数を元に、量子回路全体で必要な量子ビットの数を計測する。また、Brickwork state における非隣接 CCNOT ゲートの実装は行っていないため、以下の図 5.1 に示す図 2.1 の等価回路を作成し、この回路を元にして MBQC、Brickwork 双方の検証を行った。

Brickwork state におけるこの回路の解析結果を表 5.1 に示す。

表 5.1: cdkm08.brickwork の回路情報

| 項目 | 数値 | 備考 |
|---|---|---|
| xsize | 905 | 内訳は図 5.2 |
| ysize | 18 | ac 形式の回路と同一の値 |
| CZ ゲートの数 | 20340 | - |
| H ゲートの数 | 32562 | - |
| T ゲートの数 | 3085 | - |
| 合計量子ゲート数 | 72259 | - |
| 合計ステップ数 | 5426 | - |

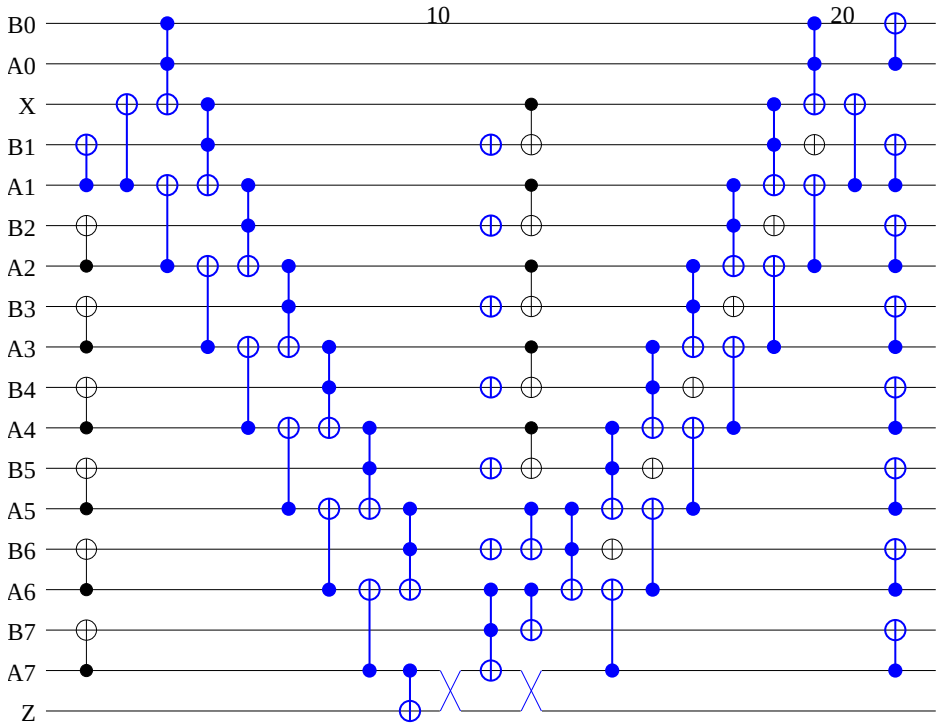


図 5.1: 量子回路の例：8 ビットの全加算回路の等価回路

合計量子ゲート数とは、回路内に含まれる全ての量子ゲートを指す。本回路において用いられている量子ゲートは Hadamard ゲート、CPhase ゲート、T ゲート、MEAS ゲートの 4 種類である。同時に生成された.brickworkhash ファイル、.brickworkangle ファイルより、各ステップにおいて必要となった量子ビットの数を図 5.2 に示す。

この Brickwork state は、これらの合計 904 列に、出力用の 1 列を加えた合計 905 列で構成される。

続いて、同じ量子回路を MBQC で解析した場合の各仕切りのサイズを図 5.4 に示す。ここで、CCNOT ゲートは以下の図 5.3 に示す量子回路 [10][11] を使用した。このゲートは $9 \times 15 = 135$ 量子ビットから構成される。また、図中の赤い丸は MEASX での測定を、橙色の丸は MEASY での測定を、そして、水色の丸は adaptive base と呼ばれる測定方法である。adaptive base とは、それまでの測定情報によって測定方法を決定する方式である。

brickworkhash、mbqchash において、共に CCNOT ゲートがボトルネックになっていることがわかる。以上より、MBQC と Brickwork state において比較可能な結果を表 5.2 に示す。

このことから、cdkm08 の回路を実行する場合、4 pitch の MBQC 回路よりも Brickwork state の方が量子ビットの数という観点から優れていることがわかる。これは、pitch の差が量子回路の大きさに関係しているためである。MBQC の pitch は 4 であるのに対し、Brickwork state の pitch は 1 である。このため、既存の MBQC モデルにおいてもより少な

```
TIME0 8
TIME1 16
TIME2 56
TIME3 56
TIME4 56
TIME5 56
TIME6 56
TIME7 56
TIME8 56
TIME9 8
TIME10 56
TIME11 8
TIME12 56
TIME13 56
TIME14 56
TIME15 56
TIME16 56
TIME17 56
TIME18 56
TIME19 16
TIME20 8
```

図 5.2: cdkm08.brickworkhash ファイルの一部

表 5.2: 可逆全加算器の量子回路：MBQC と Brickwork state の比較

| 項目 | MBQC | Brickwork state |
|---|---|---|
| xsize | 69 | 18 |
| ysize | 262 | 905 |
| 合計量子ビット | 18078 | 16290 |

い pitch のアーキテクチャを考案することで、Brickwork state よりも少ない量子ビット数で量子回路を実現することができると考えられる。
このように、AQUA プログラムの拡張によって MBQC、Brickwork state アーキテクチャにおける大規模な量子回路を実現することが可能である。これは、複数量子ゲートが存在する状況下における定性評価に相当する。ただし、現在の AQUA プログラムの字句・構文解析器、mbqctarget、brickworktarget は配列を用いた静的なメモリ確保を行っているた

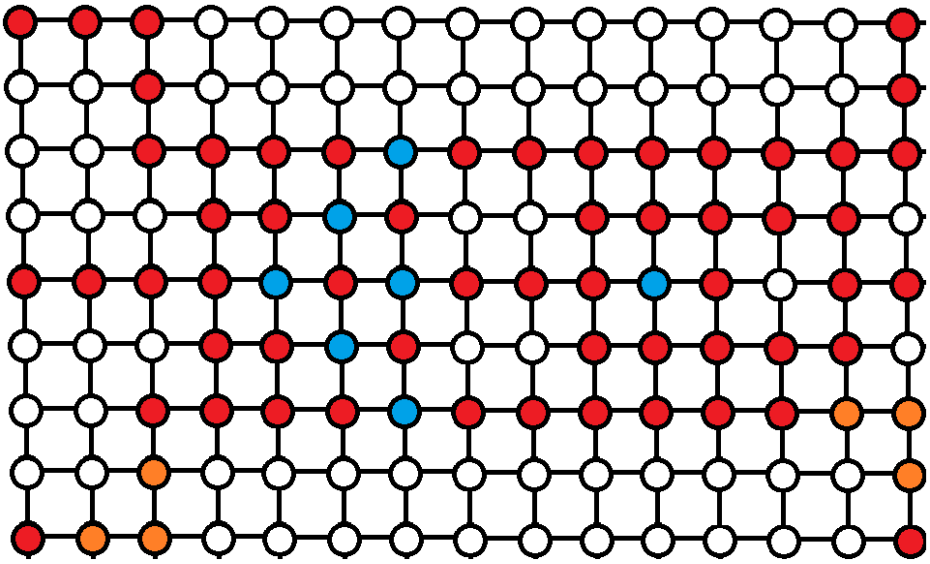

図 5.3: MBQC における CCNOT ゲート

め、実行できる回路の大きさは 10 万量子ビットまでという上限が存在する。

## 5.2 MBQC、Brickwork state アーキテクチャの比較

本節では、MBQC アーキテクチャと Brickwork state アーキテクチャを比較する。比較にあたって、目的とする量子回路に用いる量子ビット数に着目する。
表 5.3 に両アーキテクチャにおける比較可能な各量子ゲートに必要となる量子ビット数、サイズをまとめる。

表 5.3: 実装した各量子ゲートの構成

| ゲートの種類 | xsize | ysize | 合計 |
|---|---|---|---|
| Hadamard(MBQC) | 4 | 1 | 4 |
| Hadamard(Brickwork) | 4 | 1 | 4 |
| 隣接 CNOT(MBQC) | 6 | 5 | 30 |
| 隣接 CNOT(Brickwork) | 4 | 2 | 8 |
| SWAP(MBQC) | 4 | 5 | 20 |
| SWAP(Brickwork) | 4 | 2 | 8 |
| CCNOT(MBQC) | 15 | 9 | 135 |
| CCNOT(Brickwork) | 56 | 3 | 168 |

```
TIME0 6
TIME1 7
TIME2 15
TIME3 15
TIME4 15
TIME5 15
TIME6 15
TIME7 15
TIME8 15
TIME9 4
TIME10 15
TIME11 6
TIME12 15
TIME13 15
TIME14 15
TIME15 15
TIME16 15
TIME17 15
TIME18 15
TIME19 7
TIME20 6
```

図 5.4: cdkm08.mbqchash ファイルの一部

Hadamard ゲートのように、1 量子ビットに対する操作の場合は MBQC、Brickwork state アーキテクチャの間に差が存在しない。2 量子ビットに対する操作である隣接する CNOT ゲートの場合 Brickwork state が MBQC と比べて 22 量子ビット少なく済む。SWAP ゲートもまた 12 量子ビット少なく済む。一方、3 量子ビットに対する操作である CCNOT ゲートの場合は MBQC と比べて 33 量子ビット多い。これは、Brickwork state における所定の形式に量子回路を収納しないといけない性質に依る。

## 5.3 Brickwork state における SWAP ゲートに関する検証

本研究では、非隣接 CNOT ゲートに対する SWAP ゲートの配置方法を第 4.5.3 節にて 3 種類述べた。本節ではその 3 種類のモデルについて比較する。

### 5.3.1 Brickwork state における SWAP ゲートの定性評価

brickworktarget における非隣接 CNOT ゲートにおける SWAP ゲートの配置に関する挙動を確認した。各バージョンについて、確認方法等をまとめる。

- **バージョン 1**
  バージョン 1 の場合、プログラムにおける繰り返しの性質上、2 neighbor から 3 neighbor まで調べれば十分である。これらの 2 種類の非隣接 CNOT ゲートについて 5.1.1 の手法を用いて調べ、出力内容を確認した。以降、同様の方法を用いる。

- **バージョン 2**
  非隣接数が奇数か偶数かで場合分けをして実装を行い、2 neighbor から 5 neighbor まで調べた。これは、偶数の初期値を 2 neighbor、奇数の初期値を 3 neighbor、偶数の繰り返しを 4 neighbor、奇数の繰り返しを 5 neighbor としたためである。

- **バージョン 3**
  非隣接数が奇数か偶数かで場合分けをして実装を行い、2 neighbor から 6 neighbor まで調べた。これは、偶数の初期値を 4 neighbor、奇数の初期値を 3 neighbor、偶数の繰り返しを 6 neighbor、奇数の繰り返しを 5 neighbor としたためである。また、2 neighbor の場合はこの実装下において特殊であるため、バージョン 1 と同じ手法を用いた。

### 5.3.2 SWAP ゲートの配置方法による量子ビット数の比較

表 5.4 に、MBQC、および 3 種類の brickwork state における非隣接 CNOT ゲートの実装方法による量子ビット数の比較をまとめる。表中の 1 neighbor とは、隣接 CNOT ゲートのことである。また、ここで言う計算量とは、空間計算量のことであり、N neighbor 下において必要とする量子ビットの数を指す。

2 neighbor の場合、バージョン 1 から 3 まで実装方法が全て同一である。そして、2 neighbor の場合は MBQC のバージョンよりも少ない量子ビット数で実現することができる。
3 neighbor の場合、バージョン 2 が最適な実装方法である。バージョン 3 は、実装の性質上 xsize 方向に 16 pitch という大きな幅をとらなくてはならないため、小さな隣接数の下では機能しづらい。そして、バージョン 2 は隙間なく敷き詰められているため、3 neighbor において最適な配置方法と言える。
4 neighbor 以降の場合、全てバージョン 3 が優れている。これは、計算量に起因しており、

表 5.4: MBQC、Brickwork state における非隣接 CNOT ゲートの比較

| アーキテクチャ | MBQC | バージョン 1 | バージョン 2 | バージョン 3 |
|---|---|---|---|---|
| 1 neighbor | 28 | 8 | 8 | 8 |
| 2 neighbor | 63 | 36 | 36 | 36 |
| 3 neighbor | 91 | 80 | 48 | 64 |
| 4 neighbor | 119 | 140 | 100 | 80 |
| 5 neighbor | 147 | 216 | 120 | 96 |
| 6 neighbor | 175 | 308 | 196 | 112 |
| 7 neighbor | 203 | 416 | 224 | 128 |
| 8 neighbor | 231 | 540 | 324 | 144 |
| 9 neighbor | 259 | 680 | 360 | 160 |
| 10 neighbor | 287 | 836 | 484 | 176 |
| N neighbor | $28(n-2)+63$ | $(8n-4)(n+1)$ | $(8\lfloor\frac{n}{2}\rfloor+4)(n+1)$ | $16(n+1)$ |
| 計算量 | $O(28n)$ | $O(8n^2)$ | $O(4n^2)$ | $O(16n)$ |

バージョン 3 の計算量は $n$ のオーダーで済むのに対し、バージョン 1,2 の計算量は $n^2$ のオーダーを必要とするため、neighbor 数が多くなると、バージョン 3 が徐々に有利になっていく。なお、MBQC とバージョン 3 を比較した場合、初期値 (隣接する CNOT ゲート)、係数共にバージョン 3 の方が少ないため、バージョン 3 は MBQC よりも優れた手法であることが言える。
MBQC とバージョン 1 を比較した場合、4 neighbor において、MBQC とバージョン 2 を比較した場合、6 neighbor において MBQC アーキテクチャの方が必要とする量子ビット数を少なくすることができる。これらもまた、計算量のオーダーの違いによるものである。そして、バージョン 1 とバージョン 2 の間では、バージョン 2 の方が優れている。バージョン 1 は V 字型に配置しているため、かなりのスペースを必要とするが、バージョン 2 は X 字型に配置するため、その $\frac{1}{2}$ のスペースで処理を行うことが可能である。

## 5.4 考察

本研究において、最適化の対象とする回路が主に Brickwork state アーキテクチャ側であったため、上述の通りの結果となったとも言える。MBQC 側にもまた、最適化の余地はあり、手法次第では Brickwork state よりも少ない量子ビット数で量子回路を実現することができる可能性がある。また、本研究では Brickwork state においても CCNOT ゲートをはじめとする非隣接の CNOT ゲート以外の最適化によって、更に少ない量子ビット数で量子回路を実現することが可能である。
そして、Brickwork state における SWAP ゲートの実装の意義は大きい。既存の $20 \times 2$ の

モデルの場合、表 5.4 においてバージョン 1 の計算量は $40n^2$ 、バージョン 2 の計算量は $20n^2$、バージョン 3 の計算量は $80(n+1)$ となり、全ての非隣接モデルにおいて、MBQC アーキテクチャよりも必要となる量子ビット数が多くなってしまう。このため、$4 \times 2$ のサイズの SWAP ゲートの実装は、Brickwork state において効率よく量子ビットを扱うために欠かせない要素である。
本研究において用いた量子回路を更に最適化する場合、複数ゲート間における回路の結合、配置の最適化が有力である。入力する回路を MBQC、Brickwork state に特化した等価回路に変換した上でそれぞれのアーキテクチャにおける量子回路を生成することで、より一層、量子ビット数の最適化を実現することができる。

# 第6章　結論

## 6.1　まとめ

より良いクラウドサービスを構築していく中で、秘密計算はその安全性を大幅に向上させる。本研究では、秘密計算の効率を改善することができると期待されている量子コンピューティングを用いる秘密量子計算に着目した。そして、秘密量子計算の一連の流れの中で、最初の一歩である入力される量子回路を変換し、サーバに送信する関数の内容を生成するプログラム、およびそのコードを作成した。
実装したプログラムのうち、Bickwork state アーキテクチャを元にした brickworktarget プログラムの量子回路の組は機能的完全性を満たす。これは、任意の回路を等価回路に変換することができることを意味する。MBQC アーキテクチャを元にした mbqctarget プログラムについても、CCNOT ゲート、もしくは T ゲートを実装することで機能的完全性を満たすことができる。本研究で用いた MBQC アーキテクチャの測定方法は X 軸方向の測定、Y 軸方向の測定に限定しているため、T ゲートの $\frac{\pi}{4}$ 回転を元にした測定を行えないことが要因である。この測定手法、あるいは T ゲートを実装することで、MBQC アーキテクチャもまた機能的完全性を満たすことができる。
また、既存の MBQC アーキテクチャよりも Brickwork state アーキテクチャの方が量子ビットの数を最適化することができるかどうかの検証を行った。主に対象とするゲートは、非隣接 CNOT ゲートであり、SWAP ゲートを活用することで既存の隣接 CNOT ゲートに変換可能である。そして、非隣接 CNOT ゲートにおいて、部分的に Brickwork state の方が MBQC よりも効率よく量子ビットを扱うことができる方法を発見した。最適化の手法は brickworktarget プログラム内に組み込まれている。更に効率の良い方法が見つかった場合、該当部分を改変、拡張することでより少ない量子ビット数で量子回路を実現することが可能である。
生成されたコードは、定められた形式暗号化を施し、サーバに送信することで、秘密量子計算の一部分である暗号化された関数の生成に役立てられる。測定の際に利用する角度の情報の一覧を出力するファイル (brickworkangle ファイル) に従って量子コンピューティングサーバの所定の位置の量子ビットを測定し、正しく復号することで Brickwork state の演算を行うことができる。
本研究は、既存の ac アーキテクチャから MBQC、Brickwork state アーキテクチャへと変換することができる初めてのコンパイラである。また、本研究の成果である aqua-tools(AQUA プログラム) は、aqua-cat サーバの svn リポジトリにて公開する予定である。

## 6.2 今後の展望

### 6.2.1 ユニバーサル秘密量子計算シミュレータ

ユニバーサル秘密量子計算を実現するにあたって、その動作を模倣したシミュレータを考える。このシミュレータは古典サーバ上で動作し、ユニバーサル秘密量子計算の内容をシミュレートする。量子コンピューティングのアーキテクチャとして、Brickwork state を用いる。これは、理論面からユニバーサル秘密量子計算が盗聴が不可能であること、理論的に各 Brickwork state の量子回路が動作することを示すのに有力なシミュレータとなる。理論面で活躍するこのシミュレータは、複雑で大きな行列計算をする必要がある。
このシミュレータを実行するにあたって、本研究で用いた brickworktarget プログラムを活用できる。まず、生成された量子回路を動作検証に用いることができる。この量子回路をシミュレートする場合、まず、元となる.ac 形式の回路、および.brickwork 形式の量子回路を量子計算シミュレータで実行し、双方の結果が一致するかどうか確かめることで、理論的に動作が保証される。これにより、小規模な回路の理論的な動作保証は可能であり、それらを組み合わせることで大規模な量子コンピュータを用いた動作実験を確実なものにできる。この中に、暗号化の処理を入れることで、秘密量子計算において盗聴が可能かどうかの理論的な検証を行うことも可能である。

### 6.2.2 ユニバーサル秘密量子計算の実現に向けて

ユニバーサル秘密量子計算は図 3.1.2 に示す手法で実現する。本研究は図中の黄色の枠で囲われた部分の実装にあたる。その先のシステムが今後に実装され、ユニバーサル秘密量子計算が実現すると、秘密量子計算は中央集権型のシステムになると考えられる。中央集権型のシステムとは、要所に大規模な量子コンピュータを配置し、各端末からアクセスするクライアント・サーバ方式のシステムである。このシステムを構築するにあたって必要となる要素として、大規模な量子ネットワークの整備、量子コンピューティングの構築、そして精度の改善、量子エラー訂正の構築などが挙げられる。
これらの問題を解決し、システムの構築が完成すると、ユニバーサル秘密量子計算は量子コンピュータを用いたビジネスモデルとして活躍する。量子コンピュータのメリットとして現在考えられているのは、RSA 暗号の解読をはじめとする研究目的の用途が多いが、ユニバーサル秘密量子計算は機能的完全性を備えており、可逆計算を元にした様々な演算を実行することができる。これにより、今まででは実現できなかった超高速な処理が実現する可能性を秘めている。

## 6.3 今後の課題

今後は、以下の方向性で活動を進めていく。

- **AQUA 言語の拡張**
  AQUA 言語は、様々な改善できる余地がある。例えば、パウリ行列の 1 つである Z ゲートの追加や、構文解析に繰り返しを定義することによるコード量の最適化などである。AQUA 言語が高級言語に対応できるようにすることは、プログラムをより行いやすくする上で重要である。また、AQUA プログラムは静的にメモリを確保する部分が多い。これを改善することで、使用するメモリの量を最適化することが可能である。

- **MBQC 形式、Brickwork state 形式の拡張**
  本研究で多くの図として用いた MBQC や Brickwork state における量子ビットの相互作用関係図を出力するプログラムが存在すれば便利である。mbqctarget、brickworktarget を拡張し、視覚的な MBQC、Brickwork state の図を生成することは研究を進めていく上で大きな利便性をもつ。
  また、最適化方面でも可能性は多く残されている。MBQC アーキテクチャ、Brickwork state アーキテクチャにおける複数量子ゲート、量子ビット下における最適化は大規模な回路を実行する上で有力となる。
  また、AQUA プログラム下の MBQC アーキテクチャにおける CCNOT ゲートの実装による機能的完全性の充足も重要である。MBQC アーキテクチャにおいて、CCNOT ゲートを実装するのが大変な要因は、その量子ビットの多さだけでなく、adaptive base という測定方法も要因の 1 つである。この adaptive base は MBQC アーキテクチャにおける NOT ゲート [11] にも存在するが、現在の AQUA 言語に adaptive base に対応する内容を実装することが望ましい。
  また、CCNOT ゲートの非隣接モデルを実装することも重要である。CCNOT ゲートは 3 量子ビットを対象としており、Control ビット 2 つ、Target ビット 1 つから構成されるが、これらの非隣接モデルを全て包括する場合、いくつもの場合分けが必要となってくると考えられる。

- **ユニバーサル秘密量子計算シミュレータの完成**
  本研究で対象としたのは、ユニバーサル秘密量子計算シミュレータのうち図 3.1 中で最初の一歩の部分である。その後に続く処理を行うことで、ユニバーサル秘密量子計算のシミュレータを実現することができる。

- **エラー訂正への拡張**
  入力回路を元にしたコード生成時に、エラー訂正量子回路が加わったコードを生成することは実際の量子コンピューティングにおいて重要な位置づけとなる。

今後はMBQC、Brickwork state、秘密量子計算におけるにおけるシステムの拡張、最適化、システム検証の方面から問題に取り組み、より良いシステムを構築していく。

# 謝辞

本論文の作成にあたり、ご指導頂いた慶應義塾大学環境情報学部学部長村井 純博士、同学部教授徳田 英幸博士、同学部教授中村 修博士、同学部准教授楠本 博之博士、同学部准教授高汐 一紀博士、同学部准教授三次 仁博士、同学部准教授植原 啓介博士、同学部准教授中澤 仁博士、同学部准教授 Rodney D.Van Meter III 博士、同学部教授武田 圭史博士、同大学大学院政策・メディア研究科特認講師斉藤 賢爾博士に感謝致します。
特に、同学部准教授 Rodney D.Van Meter III 博士に重ねて感謝いたします。氏なしの大学生活、研究生活を想像することすらできません。また、研究に限らず日頃から多くのアドバイスをしていただき、本当に助かりました。氏の下で卒業できることを誇りに思います。
Oxford 大学リサーチアシスタント Clare Horsman 博士に感謝いたします。入学直後から非常に多くのアドバイス、量子コンピューティングを教えていただきました。ありがとうございました。
慶應義塾大学大学院准教授楠本 博之博士氏に感謝いたします。研究におけるアドバイスや、卒論における添削、助かりました。
慶應義塾大学大学院 政策・メディア研究科 特任講師 鈴木 茂哉氏に感謝いたします。卒論執筆、普段のサーバ構築に当たって多くのアドバイスをいただきました。
慶應義塾大学大学院博士課程永山翔太氏に感謝いたします。研究室に入ったばかりの私にプログラミング、量子コンピューティングを教えて頂き、研究の相談についても親身に乗っていただきました。
元国立台湾大学の Chia-Hung Chien 氏に感謝いたします。数多くの MBQC、Brickwork state に関する相談にくださり、研究を進める上でとても助かりました。
Luciano Aparicio 氏に感謝いたします。特に、英語力の向上に関してはとても助かりました。
研究室生活の中で様々な助言を頂いたり、目標となってくださった村井研究室卒業生である佐藤貴彦氏、Pham Tien Trung 氏、上原雄貴氏、重松邦彦氏、碓井利宣氏、中島明日香氏、三部剛義氏に感謝致します。
そして、研究室で同じグループであった石崎佳織氏、村田紘司氏、Nguyen Trung Duc 氏、水谷伊織氏、細川毅騎氏、松尾賢明氏、Maxim LeFleur 氏、Michael Wiegner 氏、Nhop-anon Thairungroj 氏に感謝致します。
慶應義塾大学大学院修士課程、山本知典氏、由井卓哉氏に感謝致します。日々のアドバイスは研究生活を行う上で良い励みになりました。
また、露木航平氏、高橋恒樹氏、小山忠氏とは、研究分野は違えど、大学生活の中で切磋琢磨できた良い仲間でした。ありがとうございました。

そして、徳田・村井・楠本・中村・高汐・重近・バンミーター・植原・三次・中澤合同研究プロジェクトの皆様に感謝致します。
最後に、大学入学から4年間に渡る研究生活で、私を支え続けてくれた家族である父、母、弟に心から感謝致します。

以上を持って、謝辞といたします。

2014年1月
福山 翔一朗

# 参考文献


[1] Nigel P Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography–PKC 2010*, pages 420–443. Springer, 2010.

[2] Whitfield Diffie and Martin Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[3] Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[4] Stefanie Barz, Elham Kashefi, Anne Broadbent, Joseph F Fitzsimons, Anton Zeilinger, and Philip Walther. Demonstration of blind quantum computing. *Science*, 335(6066):303–308, 2012.

[5] Shuntaro Takeda, Takahiro Mizuta, Maria Fuwa, Peter van Loock, and Akira Furusawa. Deterministic quantum teleportation of photonic quantum bits by a hybrid technique. *Nature*, 500(7462):315–318, 2013.

[6] Rodney Van Meter. Aqua 言語. `http://aqua.sfc.wide.ad.jp/`, 2003.

[7] Michael A. Nielsen. **量子コンピュータと量子通信**. オーム社, 2004.

[8] Rodney Van Meter, WJ Munro, Kae Nemoto, and Kohei M Itoh. Arithmetic on a distributed-memory quantum multicomputer. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 3(4):2, 2008.

[9] Steven A Cuccaro, Thomas G Draper, Samuel A Kutin, and David Petrie Moulton. A new quantum ripple-carry addition circuit. *arXiv preprint quant-ph/0410184*, 2004.

[10] Robert Raussendorf, Daniel E. Browne, and Hans J. Briegel. Measurement-based quantum computation on cluster states. *Phys. Rev. A*, 68:022312, Aug 2003.

[11] Agung Trisetyarso and Rodney Van Meter. Circuit design for a measurement-based quantum carry-lookahead adder. *International Journal of Quantum Information*, 8(05):843–867, 2010.

[12] Anne Broadbent, Joseph Fitzsimons, and Elham Kashefi. Measurement-based and universal blind quantum computation. In *Formal Methods for Quantitative Aspects of Programming Languages*, pages 43–86. Springer, 2010.

[13] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 32(4):169–178, 1978.

[14] Anne Broadbent, Joseph Fitzsimons, and Elham Kashefi. Universal blind quantum computation. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 517–526. IEEE, 2009.

[15] 中田育男. **コンパイラの構成と最適化**. 朝倉書店, 2 edition, 2009.

[16] flex: The fast lexical analyzer. `http://flex.sourceforge.net/`, 2008.

[17] Bison - gnu parser generator. `http://www.gnu.org/software/bison/`, 2012.

[18] Chia-Hung Chien, Rodney Van Meter, and Sy-Yen Kuo. Fault-tolerant operations for universal blind quantum computation. *arXiv preprint arXiv:1306.3664*, 2013.

# 付 録A　AQUA 言語上で実装した MBQC/Brickwork stateの量子ゲート一覧

## A.1　AQUA プログラムによる出力結果：MBQC 形式

mbqc 形式から fig 形式に変換した量子回路図のうち、第**??**節、および第**??**節では取り上げなかった回路について掲載する。なお、各量子ゲートの元の量子回路（AC 形式で記載された回路）については、第 2.2 節にて掲載したものである。なお、MBQC についてはピッチがいくつか存在し、本研究の MBQC として扱うモデルはピッチが 4 の場合である。

### A.1.1　MBQC における量子ゲートの特徴

本研究で実装した MBQC における各量子ゲートの特徴を表 A.1 に示す。

表 A.1: 実装した各量子ゲートの構成

| ゲート | Hadamard | CNOT(隣接) | SWAP | I gate |
|---|---|---|---|---|
| xsize | 4 | 6 | 4 | 1 |
| ysize | 1 | 5 | 5 | 1 |
| CZ ゲートの数 | 4 | 18 | 17 | 1 |
| MEASX ゲートの数 | 1 | 9 | 14 | 1 |
| MEASY ゲートの数 | 3 | 8 | 0 | 0 |
| 合計量子ゲート数 | 12 | 65 | 51 | 3 |
| 合計ステップ数 | 6 | 14 | 13 | - |
| mbqchash 上の表記 | HAD | CNO,CNOT | SWP | III |

特に、CNOT ゲートについては非隣接の場合についても実装を行った。以下の表 A.2 にその構成をまとめる。ただし、非隣接数が n の場合は、n が十分に大きい場合と仮定した見積もりであり、n が 3 以上の場合に成り立つ。

表 A.2: 実装した非隣接 CNOT ゲートの構成

| 非隣接数 | 2 | 3 | 4 | n |
|---|---|---|---|---|
| xsize | 7 | 7 | 7 | 7 |
| ysize | 9 | 13 | 17 | $4n+1$ |
| CZ ゲートの数 | 33 | 57 | 78 | $21n-6$ |
| MEASX ゲートの数 | 24 | 41 | 58 | $17n-10$ |
| MEASY ゲートの数 | 8 | 8 | 8 | 8 |
| 合計量子ゲート数 | 137 | 210 | 280 | $66n-1$ |
| 合計ステップ数 | 28 | 52 | 73 | $21n-11$ |

SWAP ゲートについては、実装にあたって第節で述べたモデルを改変し、以下の図 A.1 に示す量子ビットの形にした。図 A.2 はその量子ビット間における相互作用を示す。なお、このゲートは全て MEASX で測定され、緑枠以外の部分は I ゲートに相当する。

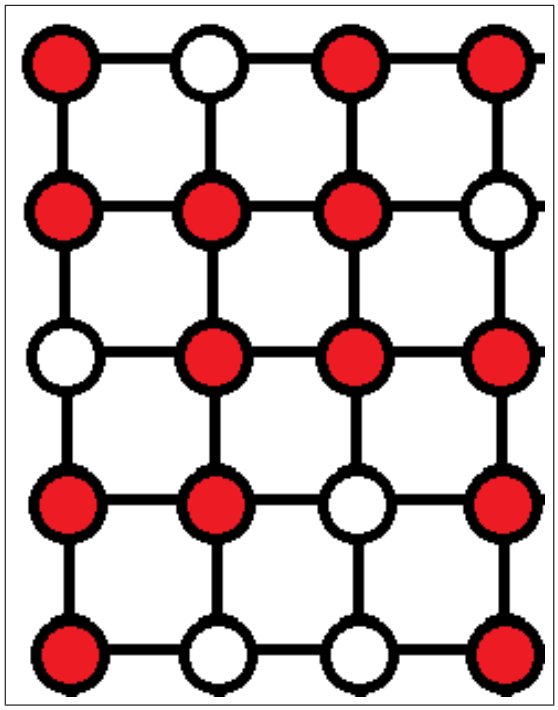

図 A.1: 実装した SWAP ゲート:MBQC モデル

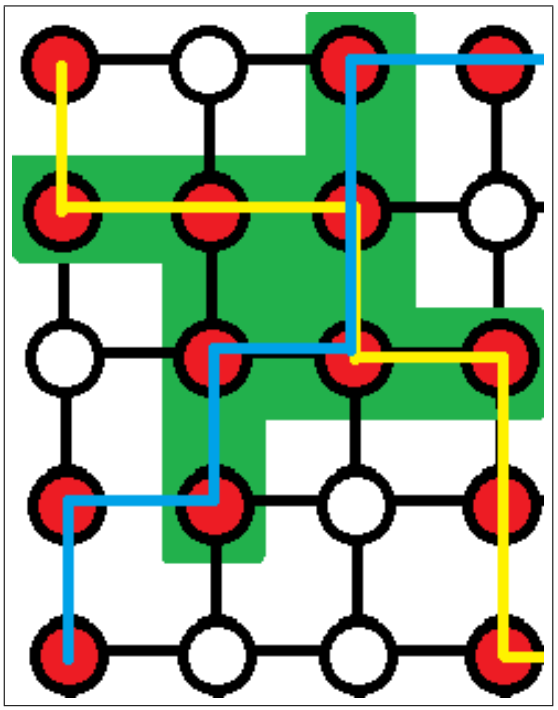

図 A.2: 実装した SWAP ゲートの相互作用

### A.1.2　MBQC における各量子ゲートの fig 出力

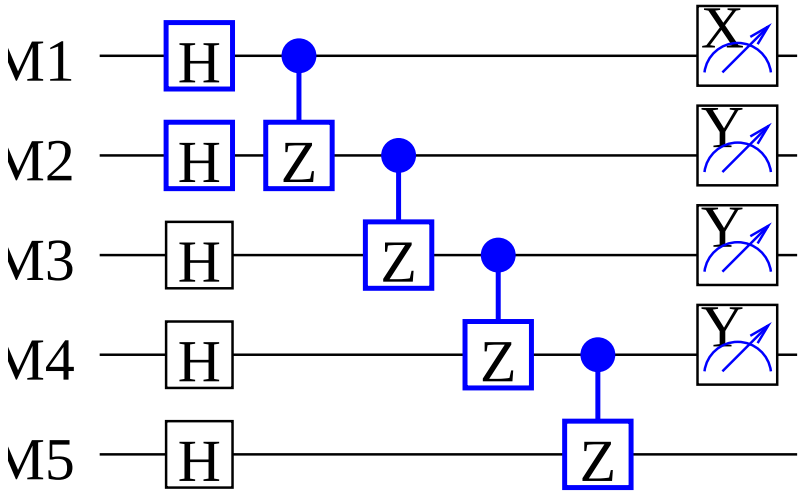


図 A.3: Hadamard ゲートの出力：MBQC 版

ここでは、3 種類の非隣接ゲートを示す。まずは、2 量子ビット離れた CNOT の場合であるが、一般的なゲートの場合は図 2.8 の通りである。そして、MBQC に変換したモデルが図 A.6 となる。更に、3 量子ビット、4 量子ビット離れた場合についても、図 A.6、図 A.7 に示す。

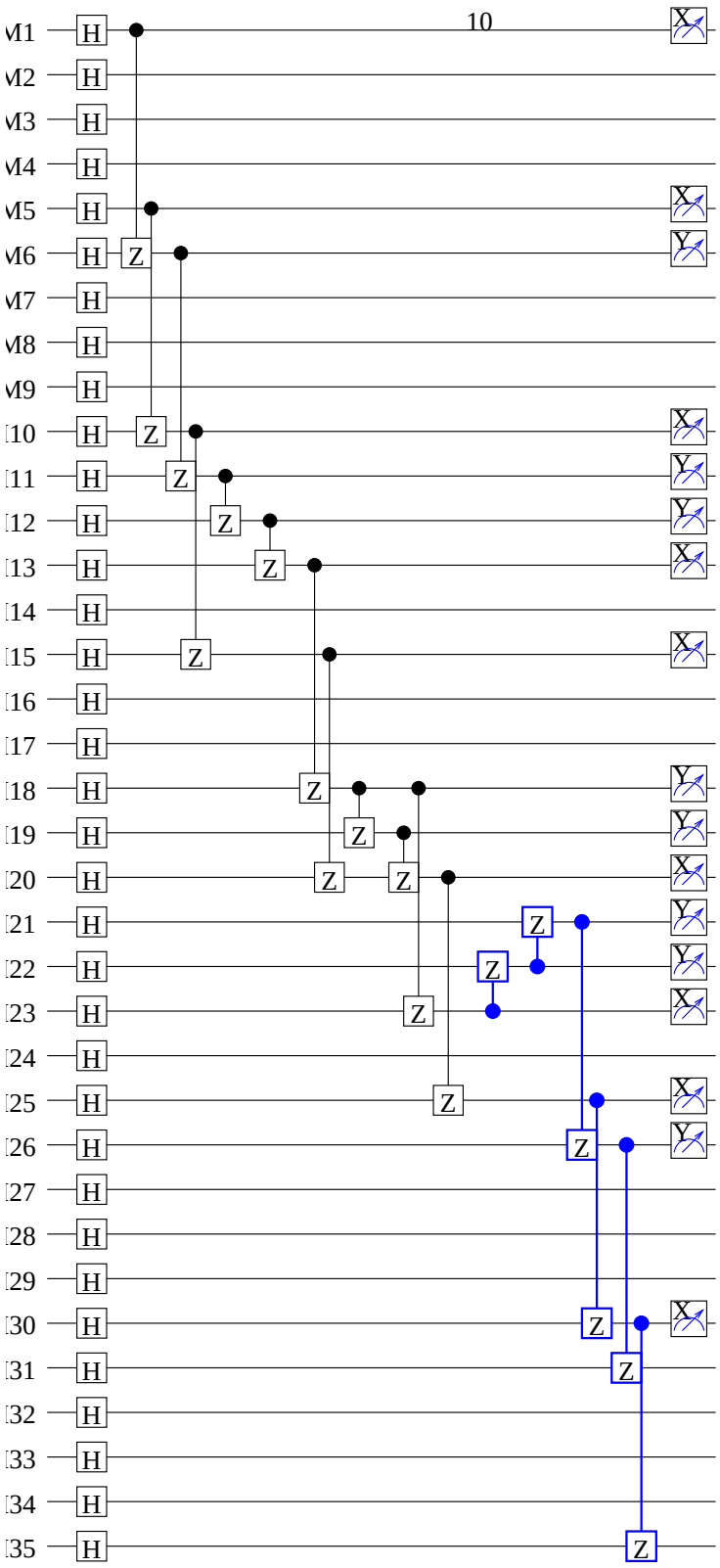


図 A.4: CNOT ゲートの出力：MBQC 版

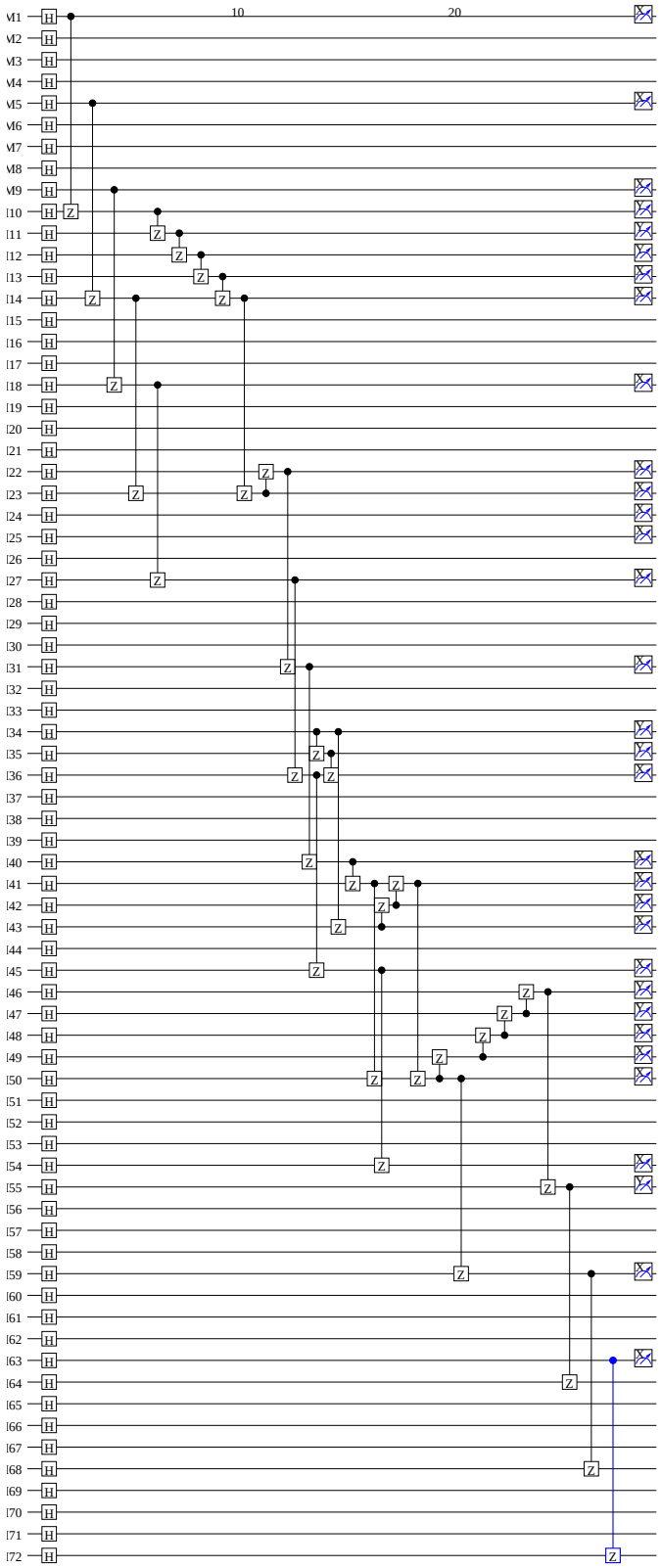


図 A.5: 非隣接ゲートの出力例（2 量子ビット離れた CNOT ゲート）：MBQC 版

図 A.6: 非隣接ゲートの出力例（3 量子ビット離れた CNOT ゲート）：MBQC 版

図 A.7: 非隣接ゲートの出力例（4 量子ビット離れた CNOT ゲート）：MBQC 版

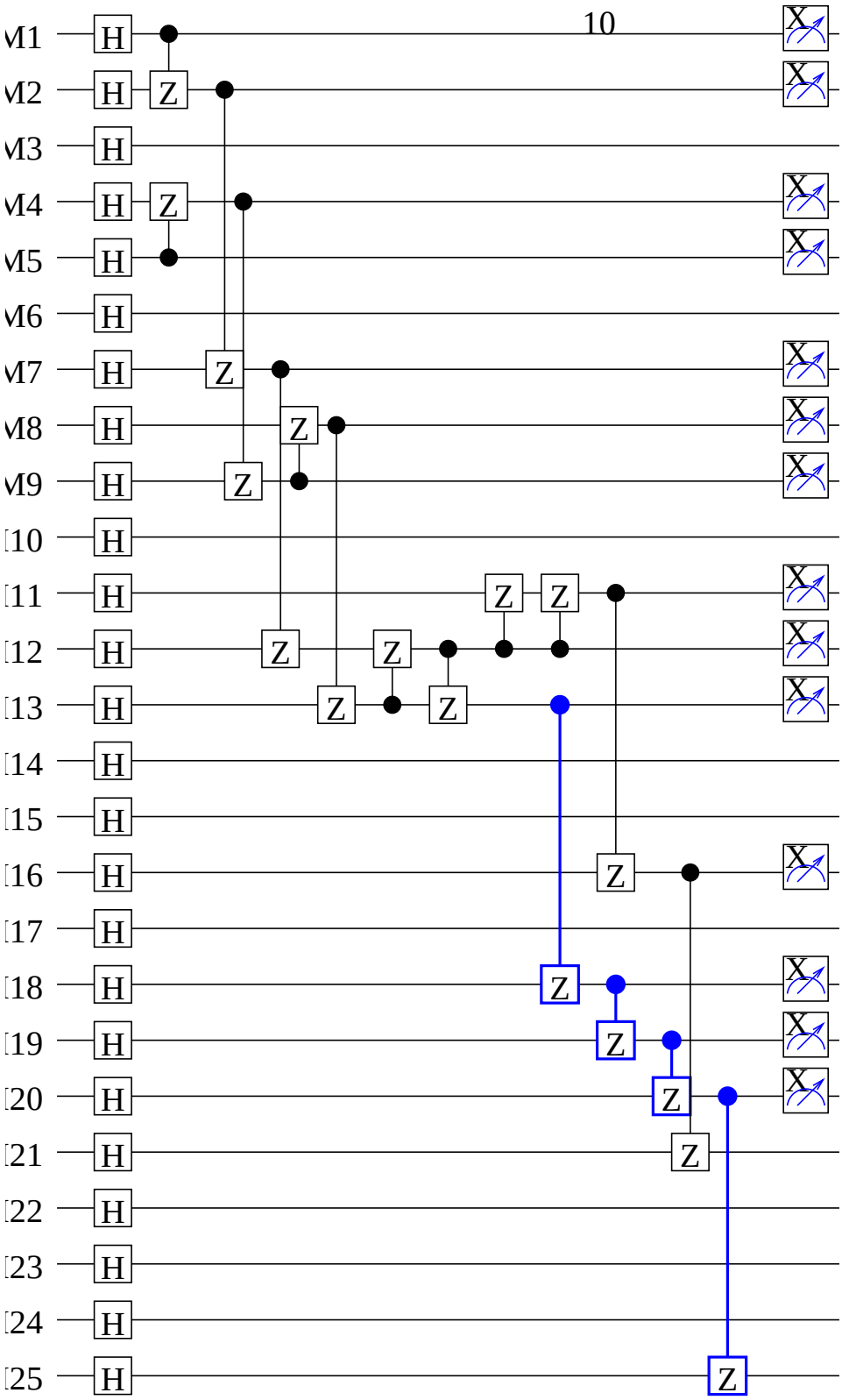


図 A.8: SWAP ゲートの出力：MBQC 版

## A.2 AQUA プログラムによる出力結果：Brickwork state 形式

### A.2.1 Brickwork state における量子ゲートの特徴

次に、Brickwork 形式で出力された第**??**節では取り上げなかった回路について紹介する。ここで、Brickworkstate のピッチは 1 であるため、通常の量子ビットと Brickwork 内における論理量子ビットの数は等しい。

表 A.3: 実装した各量子ゲートの構成 (1)

| ゲート | Hadamard | CNOT(隣接) | SWAP | CCNOT | T | S | I gate |
|---|---|---|---|---|---|---|---|
| xsize | 4 | 4 | 4 | 56 | 4 | 4 | - |
| ysize | 1 | 2 | 2 | 3 | 1 | 1 | - |
| CZ ゲートの数 | 4 | 10 | 10 | 210 | 4 | 4 | - |
| T ゲートの数 | 6 | 10 | 22 | 137 | 1 | 2 | - |
| 合計量子ゲート数 | 24 | 36 | 48 | 383 | 13 | 14 | - |
| 合計ステップ数 | 19 | 25 | 29 | 326 | 14 | 15 | - |
| brickworkhash 上の表記 | HAD | CNO,CNOT | SWP | CCN | TGA | SGA | III |

CCNOT ゲートは図の等価回路 [18] を元にして作成した。Brickwork state における CCNOT ゲートの量子回路図は、`http://web.sfc.wide.ad.jp/~fukuyama/ccnot.brickwork.gif` に示す。

### A.2.2 Brickwork state における各量子ゲートの fig 出力

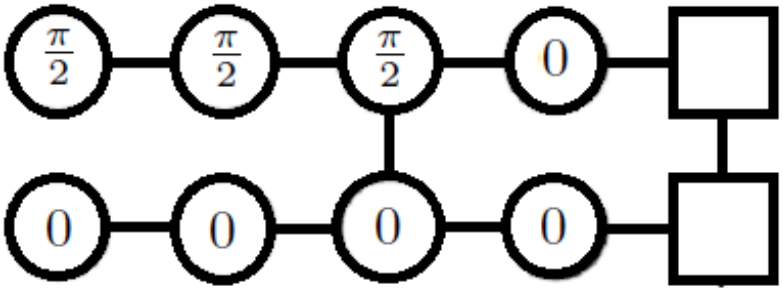


図 A.9: Hadamard ゲート：Brickwork 版

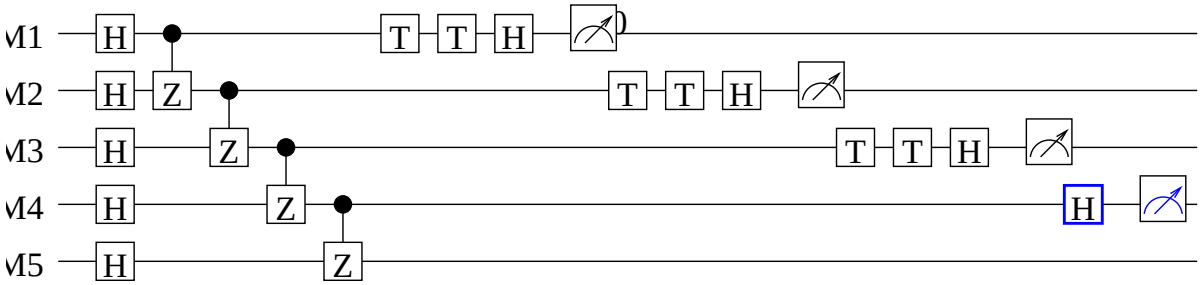


図 A.10: Hadamard ゲートの出力：Brickwork 版

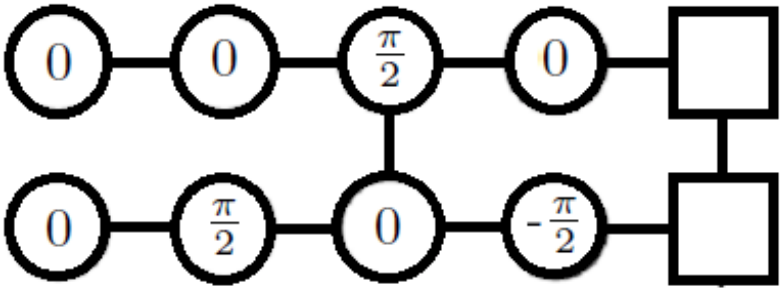


図 A.11: CNOT ゲート：Brickwork state 版

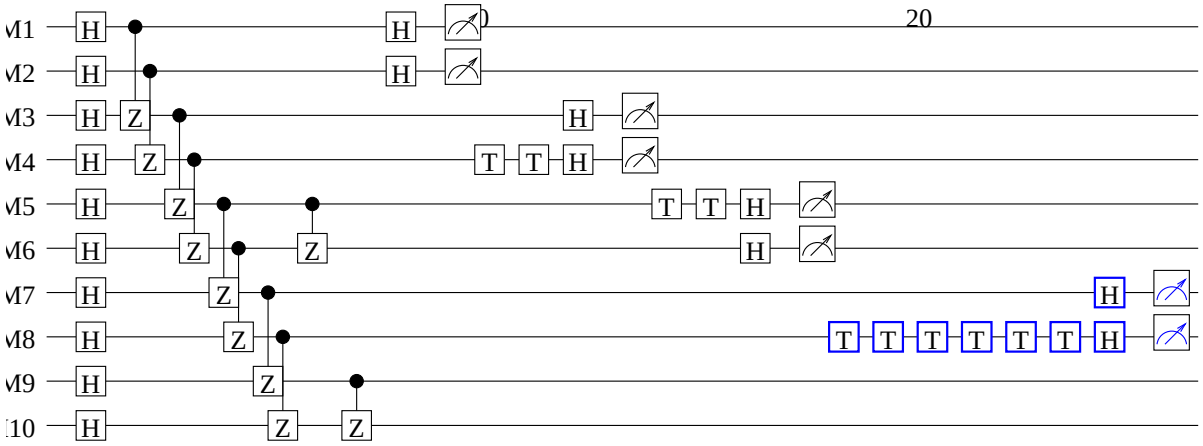

図 A.12: CNOT ゲートの出力：Brickwork state 版

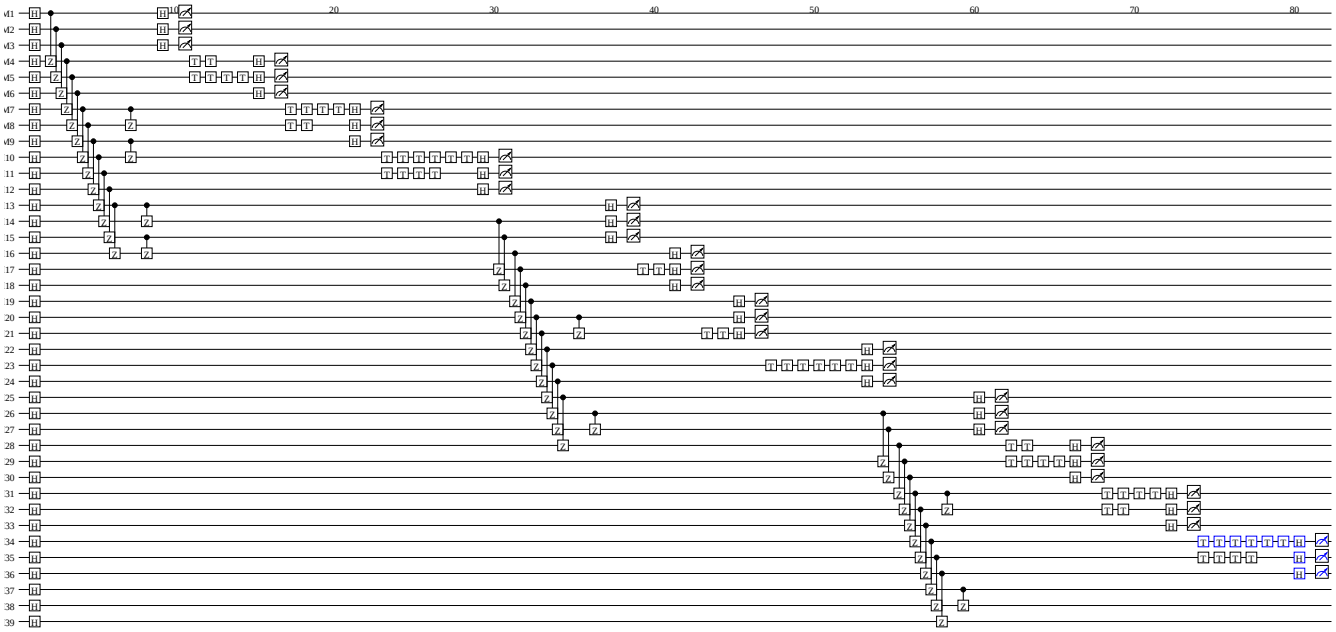

図 A.13: 非隣接ゲートの出力（2 量子ビット離れた CNOT ゲート）Brickwork state 版

図 A.14: 非隣接ゲートの出力（3 量子ビット離れた CNOT ゲート）Brickwork 版

図 A.15: 非隣接ゲートの出力（4 量子ビット離れた CNOT ゲート）Brickwork 版

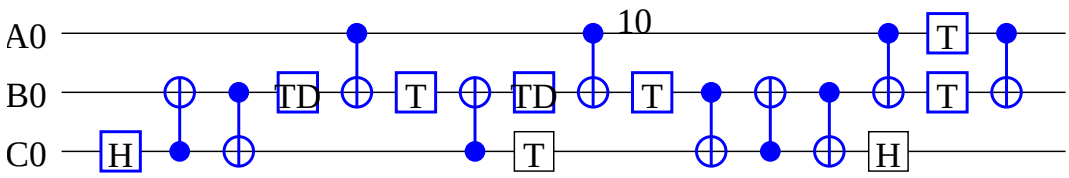


図 A.16: CCNOT ゲートにおいて利用した等価回路

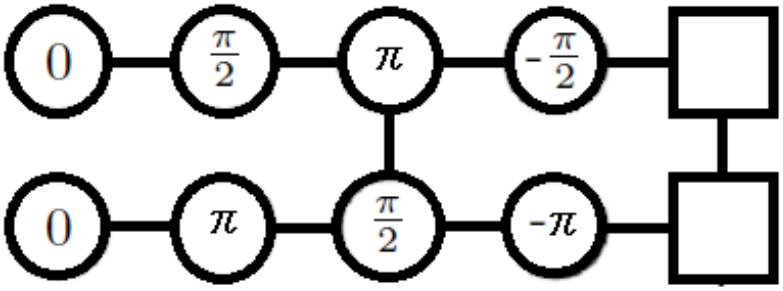


図 A.17: SWAP ゲート：Brickwork 版

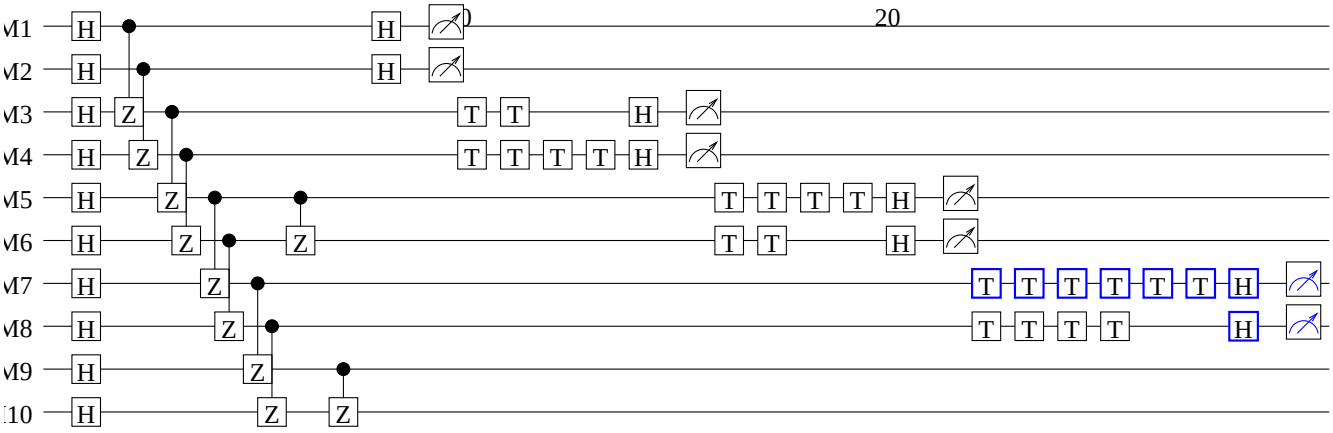


図 A.18: SWAP ゲートの出力例：Brickwork 版

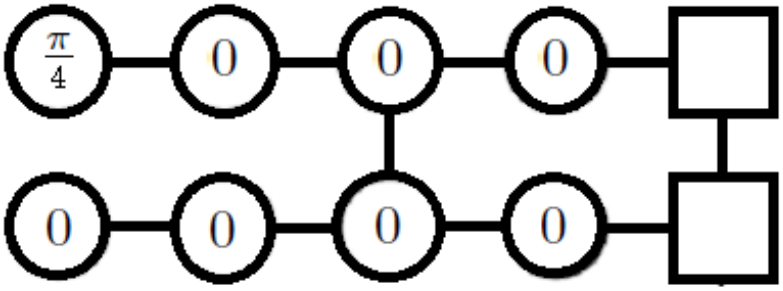


図 A.19: T ゲート：Brickwork 版

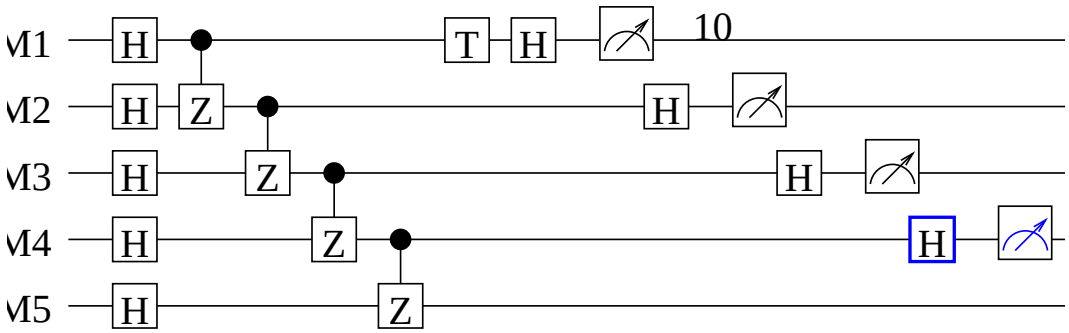


図 A.20: T ゲートの出力例：Brickwork 版

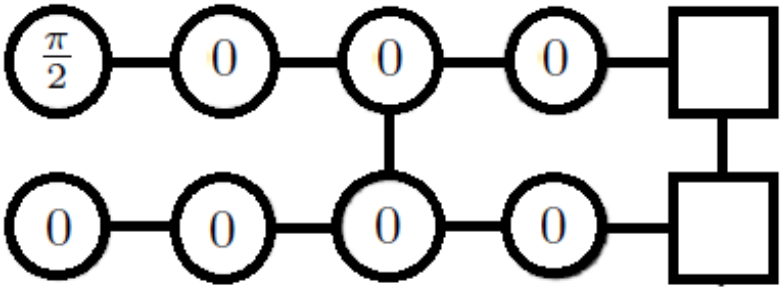


図 A.21: S ゲート：Brickwork 版

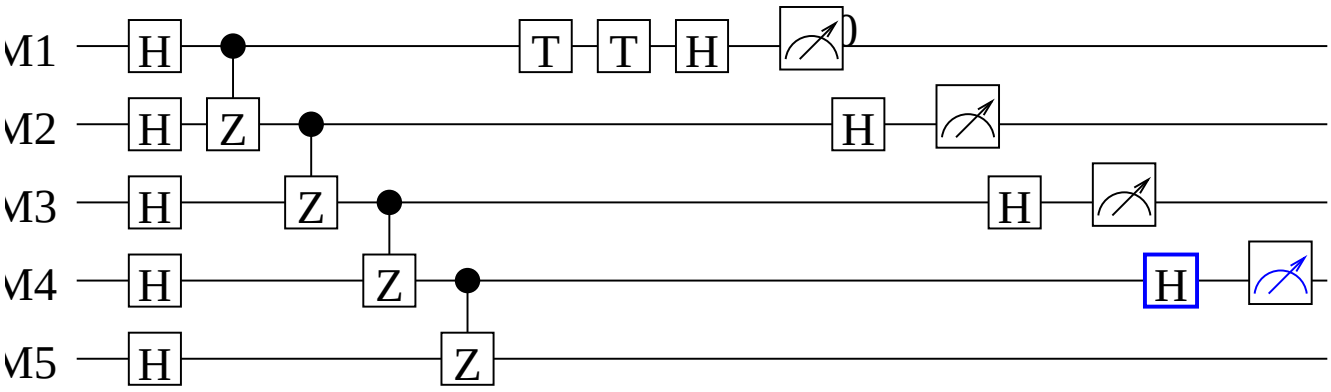


図 A.22: S ゲートの出力例：Brickwork 版