On Encoding Matrices using Quantum Circuits

Liron Mor Yosef* Haim Avron†

November 11, 2025

Abstract

Over a decade ago, it was demonstrated that quantum computing has the potential to revolutionize numerical linear algebra by enabling algorithms with complexity superior to what is classically achievable, e.g., the seminal HHL algorithm for solving linear systems. Efficient execution of such algorithms critically depends on representing inputs (matrices and vectors) as quantum circuits that encode or implement these inputs. For that task, two common circuit representations emerged in the literature: block encodings and state preparation circuits. In this paper, we systematically study encodings matrices in the form of block encodings and state preparation circuits. We examine methods for constructing these representations from matrices given in classical form, as well as quantum two-way conversions between circuit representations. Two key results we establish (among others) are: (a) a general method for efficiently constructing a block encoding of an arbitrary matrix given in classical form (entries stored in classical random access memory); and (b) low-overhead, bidirectional conversion algorithms between block encodings and state preparation circuits, showing that these models are essentially equivalent. From a technical perspective, two central components of our constructions are: (i) a special constant-depth multiplexer that simultaneously multiplexes all higher-order Pauli matrices of a given size, and (ii) an algorithm for performing a quantum conversion between a matrix's expansion in the standard basis and its expansion in the basis of higher-order Pauli matrices.

1 Introduction

Following breakthroughs such as the Harrow–Hassidim–Lloyd (HHL) algorithm [27] and subsequent works [14, 21, 53, 3, 37], numerical linear algebra has emerged as an area in which quantum computing might deliver substantial improvements over classical computing. Such quantum algorithms, which we refer to as quantum numerical linear algebra algorithms, exploit the ability of quantum computers to efficiently represent and manipulate high-dimensional vectors and matrices using exponentially fewer physical resources than is possible in classical numerical linear algebra. However, the success of quantum numerical linear algebra algorithms crucially depends on how input numerical arrays (such as matrices and vectors) are encoded into quantum circuits. Efficient representation of matrices in quantum computing is therefore not a secondary concern, but a central component that determines both the theoretical and practical feasibility of quantum numerical linear algebra.

Two main paradigms have been developed for representing numerical arrays within quantum circuits. The first, block encoding [21, 15, 11], embeds a scaled version of a matrix as a sub-block of a larger unitary operator. Block encodings forms the standard input model for many modern quantum numerical linear algebra algorithms [12, 41, 56]. The second paradigm is the use of state preparation circuits (aka amplitude encoding) [44, 51, 46, 39, 5, 4, 23, 60], which encode the entries of a matrix as probability amplitudes of

*Tel Aviv University, lm2@mail.tau.ac.il
†Tel Aviv University, haimav@tauex.tau.ac.il

quantum states. Typically, state preparation circuits are used to represent input vectors, e.g., the right hand side in HHL [27]. Nevertheless, in our previous work [42], we explored the use state preparation circuits to encode matrices, developed a limited yet useful library of matrix computational kernels on such circuits, and explored applications in the context of estimating multivariate traces and spectral sums. Although both models can represent matrices for quantum computation, state preparation circuits are often simpler to construct than block encodings, which typically require both PREPARE and SELECT oracles and can incur data loading costs that dominate the runtime [38, 55, 15, 59].

Despite close conceptual relationship between the aforementioned matrix encoding methods, no general and efficient algorithm has previously existed for translating between these two representations. This limitation prevents quantum algorithms from flexibly combining algebraic and computational tools associated with each model. Establishing such an algorithm serves not only clarify the theoretical connection between them, but also enables hybrid algorithmic designs that can exploit whichever representation is most natural for a given problem instance.

In this paper, we systematically study encoding matrices in the form of block encodings and state preparation circuits. We examine methods for constructing these representations from matrices given in classical form, as well as quantum two-way conversions between circuit representations. Two key results we establish (among others) are: (a) a general method for efficiently constructing a block encoding of an arbitrary matrix given in classical form (entries stored in classical random access memory); and (b) a low-overhead, bidirectional conversion between block encodings and state preparation circuits, showing that these models are essentially equivalent. Our previous work [42, Section 3.2.2] partially addressed conversion from block encoding to matrix state preparation, establishing that exact block encodings can be converted via a simple circuit transformation. In this paper we complete the analysis by addressing arbitrary block encodings and introducing the dual conversion: from state preparation to block encoding. The forward conversion keeps the conceptual simplicity of the exact block encoding conversion in [42, Section 3.2.2], while the reverse conversion requires substantially more complex circuits and additional qubits.

From a technical perspective, two central components of our constructions are: (i) a special constant-depth multiplexer that simultaneously multiplexes all higher-order Pauli matrices of a given size, and (ii) an algorithm for performing a quantum conversion between a matrix's expansion in the standard basis and its expansion in the basis of higher-order Pauli matrices.

The remainder of the paper is organized as follows. Section 2 introduces mathematical preliminaries and notation. Section 3 discuss both general construction of mulitplexers, and ones that are specific to rotation mulitplexers. Section 4 presents classical and quantum conversion techniques between standard and Pauli bases. Section 5 discusses how to multiplex higher order Pauli matrices. Section 6 presents algorithms for directly constructing block-encoding circuits from classical matrix representations, and Section 7 discuss algorithms for moving between representations, both from classical to quantum and from quantum to quantum (of different types).

1.1 Summary of contributions

Since we systematically study the two aforementioned matrix encodings schemes, our paper has many contributions, some of which we highlighted previously. Here we summarize all the contributions of the paper. See Table 2 for a summary of algorithmic (classical and quantum) complexities.

1. Efficient conversion between the standard and Pauli basis representation of a matrix, both in a classical algorithm (Subsection 4.2) and quantumly in the form of converting an encoding in one basis to the other (Subsection 4.3, Theorem 25). Conversions are two-way.

2. Efficient algorithms for building block encodings of matrices given their classical representation either in the standard (Subsection 6.3) or the Pauli (Subsection 6.2) basis. Our algorithms produce circuits with lower depth than all previously published algorithms with number of qubits that is logarithmic in

the matrix size. Is some cases, it produces block encoding with superior scale factor than all the one produced by previous algorithms.

3. Efficient algorithms for building of block encodings of matrices given as state preparation circuits either in the standard or the Pauli basis. Both conversion algorithms are reported in Subsection 7.2.2.

4. Efficient algorithm for building a matrix state preparation circuit given a block encoding of the matrix (Subsection 7.2.1).

5. Pauli Multiplexer Circuit ($\mathcal{PMX}_n$): we design and analyze an ultra-efficient quantum circuit that multiplexes the complete set of n-qubit higher-order Pauli operators. The $\mathcal{PMX}_n$ circuit achieves constant gate depth and a T-depth of one, independent of n, with a logarithmic gate count that scales as $O(n)$. This efficiency arises from an ultra-sparse structure in the multiplexer's rotation angles after a Walsh–Hadamard transform. See Section 5. This technical contribution is necessary for our conversion algorithms, but might be of independent interest.

1.2 Related work

Many quantum algorithms require encoding non-unitary matrices in quantum circuits. Block encoding addresses this challenge by embedding arbitrary matrices within larger unitary operators, enabling their use in quantum circuits. This technique, combined with quantum singular value transformation (QSVT), provides a framework for performing various linear algebra operations on quantum computers [12, 21, 56, 41].

Block encoding represents a scaled matrix $\mathbf{A}/\alpha \in \mathbb{C}^{N\times N}$ by embedding it within a larger unitary matrix $\mathbf{U} \in \mathbb{C}^{2^p N\times 2^p N}$, formally expressed as $\mathbf{A} = (\langle 0| \otimes \mathbf{I}_{2^p})\mathbf{U}(|0\rangle \otimes \mathbf{I}_{2^p})$ where $\mathbf{I}_{2^p}$ is the $2^p \times 2^p$ identity matrix for p ancillary qubits. However, efficient construction of block encodings is challenging, with works advocating the use of resources like QRAM [22, 15]. Related is the Linear Combination of Unitaries (LCU) framework which is a central and widely-used strategy for synthesizing block encodings [32, 8].

Early works on synthesizing block encodings mainly addressed construction of block encodings for sparse matrices. Childs and Kothari developed a pioneering technique for simulating sparse Hamiltonians, which was later adapted for block encoding applications [13]. Subsequent works introduced powerful techniques based on quantum walks and qubitization [8, 38]. Later, the introduction of QSVT provided a unifying framework that uses block encodings to perform advanced matrix arithmetic, such as inversion and filtering [21]. Crucially, these foundational algorithms are typically analyzed in the oracle model, and their efficiency is measured in query complexity which is the number of calls to an oracle that provides information about the matrix's non-zero entries. This abstract complexity model does not directly translate to the explicit gate-level circuit costs that are the focus of our work.

As for gate-level circuit constructions, Camps et al. show how to build block encoding circuits for structured sparse matrices [10]. For a scaled s-sparse matrix $\mathbf{A}$ ($\|\mathbf{A}\|_2 \leq 1$), they block-encode $\mathbf{A}/s$ using two pieces: a structured unitary that points to each nonzero entry and a value unitary that writes that entry's magnitude. For symmetric stochastic matrices $\mathbf{P}$, they avoid the $1/s$ scaling by constructing a symmetric block-encoding of $\mathbf{P}$ itself (rather than $\mathbf{P}/s$).

Several works considered block encodings of dense matrices. Camps et al. introduced FABLE: an efficient method for constructing block encoding quantum circuits with circuit depth $O(N^2)$ [11]. Their approach revolves around encoding matrix entries through angles derived from their inverse cosine, implementing these angles using $\mathcal{R}_y$ rotations with a one-dimensional multiplexer. The method applies Hadamard gates on both sides (which can be viewed as multiplication by all-ones vectors from both sides), resulting in additional scaling factor of $O(1/N)$ in the block encoding's prefactor. While this scaling factor potentially requires additional measurement shots to achieve desired precision, the method's efficient circuit depth makes it attractive. Recent variants extend FABLE to sparse inputs: the S-FABLE and "Lazy" S-FABLE (LS-FABLE) schemes adapt FABLE to unstructured sparse matrices, empirically reducing resources (for matrices with with $O(N)$

nonzero entries they report roughly $O(N)$ rotations and $O(N \log N)$ CNOTs) while preserving accuracy-vs-compression trade-offs [33]. Complementary work shows how to block-encode arithmetically structured matrices directly from compact descriptions (e.g., Toeplitz, tridiagonal), reducing data-loading overheads when structure is available [55]. Chakraborty et. al. [12] developed techniques for block encoding dense matrices using QSVT, achieving query complexity $O(\kappa \text{poly}(\log N/\epsilon))$ for matrices with condition number κ. Dong et al. [17] formalized a general construction for block encoding an arbitrary matrix, assuming the ability to implement multiplexers that control rotations based on the matrix entries. Clader et al. [15] systematically analyze the circuit resources needed to block-encode dense classical matrices using QRAM, showing a minimal-depth construction with $O(N^2)$ qubits and T-depth $O(\log(N/\epsilon))$ versus a minimal-count construction with $O(N \log(1/\epsilon))$ qubits and T-depth $O(N)$. Additionally, they compare select-swap vs. bucket-brigade QRAM constructions and introduce a state preparation routine improving T-depth from $O(\log^2(N/\epsilon))$ to $O(\log(N/\epsilon))$ by utilizing ancillary qubits. As is evident from these complexities, their low depth algorithms requires a large number ancillary qubits.

State preparation (aka amplitude encoding) is another well-established method for embedding classical data vectors into quantum systems through circuits that operate on the ground state and encode vectors as probability amplitudes. Various state preparation techniques [44, 51, 6, 46, 28, 40, 23, 4] have been developed over the years and designing new ones is an active research topic. For a discussion of the state preparation literature, including the derivation and analysis of quantum circuit costs, see Section 7.1.1. In our recent work [42], we extend the concept of state preparation from vectors to matrices through matrix-state preparation, which encodes a matrix's vectorization into a quantum state. This approach enables us to develop a limited yet practical library of quantum circuits that implement fundamental linear algebra primitives, and demonstrate its utility for computing multivariate traces and spectral sums.

2 Preliminaries

This section presents the mathematical framework underlying our algorithms and their analysis. Although several definitions build on our previous work [42], we extend them here to meet the specific requirements of this paper.

2.1 Linear algebra notation

We denote scalars using Greek or Latin letters (e.g., $\alpha, \beta, \ldots$ or $x, y, \ldots$). Vectors are written in bold lowercase letters such as $\mathbf{x}, \mathbf{y}, \ldots$, matrices in bold uppercase, such as $\mathbf{A}, \mathbf{B}, \ldots$, and hypermatrices (higher dimensional arrays)[1] in calligraphic notation (e.g., $\mathcal{A}, \mathcal{B}, \ldots$)[2]. The dimension, or order, of a hypermatrix $\mathcal{A}$ is denoted by $\mathbf{order}(\mathcal{A})$. We assume 0-based indexing for vectors, matrices and hypermatrices. This is less common in the numerical linear algebra literature, but is more convenient in the context of quantum computing. We use the convention that vectors are column vectors, unless otherwise stated.

Vectorization of a hypermatrix $\mathcal{A} \in \mathbb{C}^{I_0 \times \cdots \times I_{d-1}}$ is defined recursively by

$$\mathbf{vec}(\mathcal{A}) \coloneqq \begin{cases} \mathcal{A}, & \mathbf{order}(\mathcal{A}) = 1 \\ \begin{bmatrix} \mathbf{vec}(\mathcal{A}(1, :, \ldots, :, :)) \\ \mathbf{vec}(\mathcal{A}(2, :, \ldots, :, :)) \\ \vdots \end{bmatrix}, & \text{otherwise} \end{cases}$$

[1] In numerical analysis literature, hypermatrices are usually referred to as "tensors". However, tensors are, in modern definitions, elements of a tensor product of vector spaces or multilinear functionals [36], and this definition more aligned with the use of the term "tensor" in quantum physics. Thus, we opted for the term "hypermatrix" to describe a multi-dimensional array of numbers.

[2] There is some ambiguity in the notation between hypermatrices, and subsequent notation for quantum circuits (which actually describes a tensor). We rely on context for distinguish between the two cases.

Note that for a matrix (order-2), this corresponds to row-major vectorization.

The Frobenius norm of a hypermatrix $\mathcal{A} \in \mathbb{C}^{I_0 \times \cdots \times I_{d-1}}$ is defined as

$$\|\mathcal{A}\|_F \coloneqq \sqrt{\sum_{i_0=0}^{I_0-1} \sum_{i_1=0}^{I_1-1} \cdots \sum_{i_{d-1}=0}^{I_{d-1}-1} |\mathcal{A}(i_0, \ldots, i_{d-1})|^2}$$

The definition naturally applies to matrices and vectors (order-1 hypermatrices), where it coincides with the (matrix) Frobenius norm and the 2-norm (respectively). We say that a hypermatrix $\mathcal{A}$ (respectively, matrix and vector) is normalized if $\|\mathcal{A}\|_F = 1$. The inner product between two hypermatrices $\mathcal{A}$ and $\mathcal{B}$, both of the same size, is given by $\langle \mathcal{A}, \mathcal{B} \rangle \coloneqq \mathbf{vec}(\mathcal{A})^{\mathrm{H}} \mathbf{vec}(\mathcal{B})$ where the superscript H denotes the Hermitian transpose of vectors and matrices. This is consistent with the Frobenius norm definition, i.e., $\|\mathcal{A}\|_F = \sqrt{\langle \mathcal{A}, \mathcal{A} \rangle}$.

The set of square complex matrices of size M, is a *complex* vector space of dimension M^2 . However, it is also real vector space of dimension $2M^2$. Inside that real vector space, we have the subspace of Hermitian matrices, denoted by $\mathbb{H}_M \subseteq \mathbb{C}^{M \times M}$, which consists of all Hermitian matrices of size $M \times M$. The dimension of $\mathbb{H}_M$ (over $\mathbb{R}$) is M^2 (half the dimension of $\mathbb{C}^{M \times M}$ when viewed as a real vector space, and the same as the dimension when $\mathbb{C}^{M \times M}$ is viewed as a complex vector space). On $\mathbb{H}_M$, when viewed as a real vector space, we define the following inner product, which we refer to $\mathbb{H}_M$-inner product: $\langle \mathbf{A}, \mathbf{B} \rangle_{\mathbb{H}_M} \coloneqq \mathbf{Tr}(\mathbf{AB})$. This inner product induces the Frobenius norm.

We use capital letters to denote dimension sizes, e.g., $M, N, I_1, I_2, \ldots$ Throughout the paper, whenever we encounter such dimensions we assume they are power of two, with the corresponding small letter denoting the base-2 logarithm, e.g., $n = \log_2 N, m = \log_2 M$, etc.

2.2 Matrices, circuits and quantum states: Basic notations and definitions

We use the standard state vector based formulation of quantum computing, where the system's state is represented by a unit vector in an Hilbert space. Without loss of generality, we assume that the Hilbert space is $\mathbb{C}^N$, where $N = 2^n$ for n qubits. We denote the computational basis in a n-qubit system as $|0\rangle_n, |1\rangle_n, |2\rangle_n, \ldots$. For abstract states in a n-qubit system we use the notation $|\phi\rangle_n, |\psi\rangle_n, \ldots$. We denote by $|\psi\rangle_n |\phi\rangle_m$ the $n+m$ qubit state obtained by tensoring the two states to obtain a state in $\mathbb{C}^{MN}$ where $N = 2^n$ and $M = 2^m$ with $|\phi\rangle_n$ and $|\psi\rangle_m$ being two abstract states on n and m qubits (respectively). We assume that the MSB is in the lowest index in binary expansions, e.g., $|i\rangle_n = |b_0\rangle_1 |b_1\rangle_1 \cdots |b_{n-1}\rangle_1 =: |b_0 \cdots b_{n-1}\rangle_n$ where $b_0 \cdots b_{n-1}$ is the binary expansion of i; this aligned with many physics textbooks.

We denote the n-qubit system's state whose amplitudes are given by $\boldsymbol{\alpha}$ after normalization by the ket $|\boldsymbol{\alpha}\rangle$ notation, where $\boldsymbol{\alpha} \in \mathbb{C}^N$ is the probability amplitude vector. That is, $|\boldsymbol{\alpha}\rangle \coloneqq \frac{1}{\|\boldsymbol{\alpha}\|_2} \sum_{i=0}^{n-1} \alpha_i |i\rangle_n$ where α_i is the ith entry of $\boldsymbol{\alpha}$. Note that under these conventions we have $|i\rangle_n = \left|\mathbf{e}_i^{(N)}\right\rangle$ where $\mathbf{e}_0^{(N)}, \mathbf{e}_1^{(N)}, \ldots$ are the N-dimensional identity vectors. Given two amplitude vectors $\boldsymbol{\alpha} \in \mathbb{C}^N$ and $\boldsymbol{\beta} \in \mathbb{C}^M$, we have $|\boldsymbol{\alpha}\rangle |\boldsymbol{\beta}\rangle = |\boldsymbol{\alpha} \otimes \boldsymbol{\beta}\rangle$ and this extends naturally to higher number of multiplicands.

We use calligraphic letters to denote both quantum circuits and operators on the state's Hilbert space (and, as explained in the previous subsection, hypermatrices), e.g., $\mathcal{U}$, $\mathcal{S}$ and $\mathcal{T}$, also using the same letter for a circuit and the operator it induces. For a circuit $\mathcal{U}$ on n qubits, we use $\mathbf{M}(\mathcal{U}) \in \mathbb{C}^{N \times N}$, to denote the unique unitary matrix such that for every $\boldsymbol{\alpha} \in \mathbb{C}^N$ applying $\mathcal{U}$ on the state $|\boldsymbol{\alpha}\rangle$ results in the state $|\mathbf{M}(\mathcal{U})\boldsymbol{\alpha}\rangle$. We say that a circuit $\mathcal{U}^*$ is the inverse (or adjoint) of circuit $\mathcal{U}$ if $\mathbf{M}(\mathcal{U}) = \mathbf{M}(\mathcal{U})^{\mathrm{H}}$. Given a quantum system, we denote the application of circuits sequentially with $\cdot$ or omit altogether. We say two circuits $\mathcal{U}$ and $\mathcal{V}$ on the same number of qubits are *equivalent*, denoted by $\mathcal{U} \cong \mathcal{V}$, if they induce the same unitary matrix on the Hilbert space (i.e., $\mathbf{M}(\mathcal{U}) = \mathbf{M}(\mathcal{V})$).

A quantum register is defined as contiguous subset of qubits of a multi-qubit system. The size of the register is determined by the number of qubits it encompasses. Given a system with several quantum registers, we use the standard tensor product $\otimes$ operator to denote the application of each circuit on the corresponding

register, but also use $\cdot_i$ to denote the action of a circuit $\mathcal{U}$ to only register i of the system, e.g., $\mathcal{U}\cdot_i |\boldsymbol{\alpha}\rangle$ (where the operation on the other register is the identity). We use 0-based indexing for register numbering within a system.

Given a hypermatrix $\mathcal{A} \in \mathbb{C}^{I_0 \times \cdots \times I_{d-1}}$, we denote by $||\mathcal{A}\rangle\rangle$ the state $|\mathbf{vec}(\mathcal{A})\rangle$. This state has $q = \sum_{j=0}^{d-1} i_j$ qubits, divided into d registers of size $i_0, i_1, \ldots, i_{d-1}$. We also have

$$||\mathcal{A}\rangle\rangle = \frac{1}{\|\mathcal{A}\|_F} \sum_{j_0=0}^{I_0-1} \sum_{j_1=0}^{I_1-1} \cdots \sum_{j_{k-1}=0}^{I_{d-1}-1} \mathcal{A}(j_0, \ldots, j_{d-1}) \, |j_0 \cdot I_1 I_2 \cdots I_{d-1} + j_1 \cdot I_2 I_3 \cdots I_{d-1} + \cdots + j_{d-2} I_{d-1} + j_{d-1}\rangle_q .$$

For matrices, we have

$$||\mathbf{A}\rangle\rangle = \frac{1}{\|\mathbf{A}\|_F} \sum_{j=0}^{N-1} \sum_{i=0}^{M-1} a_{i,j} \, |iN + j\rangle_{n+m}$$

where a_{ij} denotes the (i,j)th entry of $\mathbf{A}$. We use the notation $||\mathcal{A}\rangle\rangle$ (instead of $|\mathbf{vec}(\mathcal{A})\rangle$) to help the reader distinguish state descriptions that are based on hypermatrices (or matrices) as opposed to vectors.

For a hypermatrix $\mathcal{A}$ we denote the number of qubits we need to hold the state $||\mathcal{A}\rangle\rangle$ by $q(\mathcal{A})$. For a circuit $\mathcal{U}$, we denote the number of gates in $\mathcal{U}$ by $g(\mathcal{U})$, and the depth (critical path) of $\mathcal{U}$ by $d(\mathcal{U})$, and the number of qubits of $\mathcal{U}$ by $q(\mathcal{U})$.

2.3 Representing matrices using quantum circuits

In quantum numerical linear algebra, matrices are represented - typically as inputs - by quantum circuits. In the literature, the predominant paradigm for such representations is block encoding. An alternative approach uses matrix state preparation circuits. In this subsection, we describe both methods.

A *block encoding* of a matrix $\mathbf{A} \in \mathbb{C}^{N \times N}$ is any circuit for which in the corresponding unitary we have a scaled version of $\mathbf{A}$ in the top left part. The concept was formally introduced in [21], even though the idea can be traced to earlier works on Linear Combination of Unitaries (LCU) [13, 8, 38, 14]. Since its introduction, block encodings have been extensively used in the literature on quantum numerical linear algebra and its applications [12, 41].

A formal definition is as follows; the definition is, in one aspect, slightly more general than the one originally proposed in [21] in that it allows an arbitrary phase. However, it also slightly less general as it does not allow an approximation error.

Definition 1 (Block Encoding Circuit)**.** For $\alpha \geq 0$, a circuit $\mathcal{U}$ is a α-*Block Encoding* of $\mathbf{A} \in \mathbb{C}^{N \times N}$, denoted as $\mathcal{U} \in \mathbf{BE}_\alpha(\mathbf{A})$, if there exists a $\theta \in [0, 2\pi)$ such that

$$\alpha e^{\mathrm{i}\theta} \mathbf{M}(\mathcal{U}) = \begin{bmatrix} \mathbf{A} & * \\ * & * \end{bmatrix}$$

We refer to α as the *scale*. Note that the number of qubits in $\mathcal{U}$ must be at least n. Surplus qubits are called *ancillary qubits,* and their number is equal to $q(\mathcal{U}) - n$. We say that a block encoding is *exact* if there is no ancillary qubits. Note that this possible only if $\mathbf{A}$ is scaled unitary matrix.

Suppose $\mathcal{U} \in \mathbf{BE}_\alpha(\mathbf{A})$. Even if $\mathbf{A}$ is Hermitian this does not necessarily hold for $\mathbf{M}(\mathcal{U})$. However, in some cases it might be desirable for the embedding matrix to be Hermitian [17]. The following defines a more restricted form of block encoding.

Definition 2 (Hermitian Block Encoding Circuit)**.** For $\alpha \in \mathbb{R}$, a circuit $\mathcal{U}$ is a α-*Hermitian Block Encoding* of an Hermitian matrix $\mathbf{A} \in \mathbb{C}^{N \times N}$, denoted as $\mathcal{U} \in \mathbf{HBE}_\alpha(\mathbf{A})$, if

$$\alpha \mathbf{M}(\mathcal{U}) = \begin{bmatrix} \mathbf{A} & * \\ * & * \end{bmatrix}$$

and $\mathbf{M}(\mathcal{U})$ is Hermitian. We refer to $|\alpha|$ as the *scale*.

We denote block encoding circuits of a matrix $\mathbf{A}$ by $\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}}$, and $\mathcal{U}_{\mathbf{A}}^{\mathrm{HBE}}$ for a Hermitian block encodings. Let $a = q(\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}}) - n$. The definition of block encoding implies that for any unit norm vector $\boldsymbol{\psi} \in \mathbb{C}^N$, there exists a θ such that

$$\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}} |0\rangle_a |\boldsymbol{\psi}\rangle = \sqrt{p(\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}}, \boldsymbol{\psi})} e^{-\mathrm{i}\theta} |0\rangle_a |\mathbf{A}\boldsymbol{\psi}\rangle = +\sqrt{1 - p(\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}}, \boldsymbol{\psi})} |\perp\rangle_{q(\mathcal{U})}$$

where $|\perp\rangle$ satisfies $(|0^a\rangle\langle 0^a| \otimes \mathbf{I}_N) |\perp\rangle_{q(\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}})} = 0$ and

$$p(\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}}, \boldsymbol{\psi}) = \frac{\|\mathbf{A}\boldsymbol{\psi}\|_2^2}{\alpha^2} \tag{1}$$

We can now measure the ancillary qubits, and if we get $|0\rangle_a$ we have successfully prepared $|\mathbf{A}\boldsymbol{\psi}\rangle$. The probability of successfully preparing $|\mathbf{A}\boldsymbol{\psi}\rangle$ is $p(\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}}, \boldsymbol{\psi}) \geq \sigma_{\min}(\mathbf{A})^2/\alpha^2$. We see that it is desirable to for the scaling factor α to be as small possible so to maximize the success probability. The only lower bound shown in the literature for α is $\alpha \geq \|\mathbf{A}\|_2$. Another trivial to prove bound is $\alpha \geq \frac{\|\mathbf{A}\|_F}{\sqrt{M}}$. Both bounds are not informative.

Many algorithms that make extensive use of block encodings assume the input matrix is given block encodings form. Consequently, having efficient block encoding circuits for input matrices is crucial for realizing these algorithms' claimed quantum advantage. However, block encoding is inherently restricted to matrices. Furthermore, in the definition above, we defined block encodings only for square matrices. Though we could have defined it to allow rectangular matrices, such block encodings are less ubiquitous. Quantum numerical linear algebra algorithms are mostly defined only for block encoding of square matrices. For numerical arrays that are not square matrices, such as vectors, rectangular matrices or higher order tensors, it is more convenient to work with *state preparation circuits*. For example, algorithms for solving linear equations assume the right hand side is given as a state preparation circuit [27, 14, 13].

Definition 3 (Hypermatrix State Preparation Circuit)**.** Let $I_0, I_1, \ldots, I_{d-1}$ be powers of 2, and let $\mathcal{A} \in \mathbb{C}^{I_0 \times I_1 \times \cdots \times I_{d-1}}$ be a hypermatrix. For $\alpha \geq 0$, a circuit $\mathcal{U}$ is a α-*Hypermatrix State Preparation Circuit* of $\mathcal{A}$, denoted by $\mathcal{U} \in \mathbf{HS}_\alpha(\mathcal{A})$, if there exists a $\theta \in [0, 2\pi)$ such that

$$\alpha e^{\mathrm{i}\theta} \mathbf{M}(\mathcal{U}_{\mathcal{A}}) = \begin{bmatrix} \vdots & * & \cdots & * \\ \mathbf{vec}(\mathcal{A}) & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \boldsymbol{\psi} & \vdots & & \vdots \\ \vdots & * & \cdots & * \end{bmatrix},$$

for some (possibly empty) vector $\boldsymbol{\psi}$. We refer to α as the *scale*.

Suppose $\mathcal{U} \in \mathbf{HS}_\alpha(\mathcal{A})$. Since $\mathbf{M}(\mathcal{U})$ is, by definition, unitary, α and $\boldsymbol{\psi}$ must be such that first column of $\mathbf{M}(\mathcal{U})$ has unit norm. Thus, we must have $\alpha \geq \|\mathcal{A}\|_F$. While the requirement of $I_0, I_1, \ldots, I_{d-1}$ to be powers of 2 is formally redundant, it enables a more streamlined analysis of hypermatrix state preparation circuits. For example, in the above the number of elements in $\boldsymbol{\psi}$ is $2^{q(\mathcal{U})} - I_0 I_1 \cdots I_{d-1} = (2^{q(\mathcal{U}) - \sum i_j} - 1) I_0 I_1 \cdots I_{d-1}$. In the notation $\mathcal{U} \in \mathbf{HS}_\alpha(\mathcal{A})$ we do not specify the number of qubits in $\mathcal{U}$. It is property of the circuit itself, not the fact that it is a hypermatrix state preparation circuit. At the vary least, the number of qubits in $\mathcal{U}$ must be $\sum i_j$. Thus, the number of *ancillary qubits* in $\mathcal{U} \in \mathbf{HS}_\alpha(\mathcal{A})$ is $q(\mathcal{U}) - \sum i_j$. We denote hypermatrix state preparation circuits of a hypermatrix $\mathcal{A}$ as $\mathcal{U}_{\mathcal{A}}^{\mathrm{SP}}$.

Suppose that $\mathcal{U}_{\mathcal{A}}^{\mathrm{SP}} \in \mathbf{HS}_{\alpha}(\mathcal{A})$. The definition implies there exists a θ such that

$$\mathcal{U}_{\mathcal{A}}^{\mathrm{SP}} |0\rangle_{q(\mathcal{U})} = \sqrt{p(\mathcal{U}_{\mathcal{A}}^{\mathrm{SP}})} e^{-\mathrm{i}\theta} |0\rangle_{q(\mathcal{U}_{\mathcal{A}}^{\mathrm{SP}}) - \sum i_j} ||\mathcal{A}\rangle\rangle + \sqrt{1 - p(\mathcal{U}_{\mathcal{A}}^{\mathrm{SP}})} |\perp\rangle$$

where $|\perp\rangle$ satisfies the orthogonality condition $\left\langle 0^{q(\mathcal{U}_{\mathcal{A}}^{\mathrm{SP}}) - \sum i_j} \right| \otimes \mathcal{I}_{\sum i_j}) |\perp\rangle = 0$, and $p(\mathcal{U}_{\mathcal{A}}^{\mathrm{SP}}) = \frac{\|\mathcal{A}\|_F^2}{\alpha^2}$ is the probability of measuring $|0\rangle_{q(\mathcal{U}_{\mathcal{A}}^{\mathrm{SP}}) - \sum i_j}$ when measuring only the last $q(\mathcal{U}_{\mathcal{A}}^{\mathrm{SP}}) - \sum i_j$ qubits. In other words, $p(\mathcal{U}_{\mathcal{A}}^{\mathrm{SP}})$ is the probability of actually succeeding in preparing $||\mathcal{A}\rangle\rangle$, up to global phase, in the second register. Thus, it is desirable for α to be as small as possible. We must have $\alpha \geq \|\mathcal{A}\|_F$. We say that the state preparation circuit is *exact* if $\alpha = \|\mathcal{A}\|_F$ (equivalently, the probability of preparing $||\mathcal{A}\rangle\rangle$ in the target register is 1). A state preparation circuit can be exact even when there are ancillary qubits, though if there are no ancillary qubits the circuit must be exact.

Two special cases of hypermatrix state preparation circuits are worth noting, and giving them a dedicated notation. When $k = 1$, we are dealing with (vector) state preparation circuits with garbage, a construction extensively used in the literature [44, 51, 46]. We use the notation $\mathcal{U}_{\mathbf{v}}^{\mathrm{SP}} \in \mathbf{SP}_{\alpha}(\mathbf{v})$. When $k = 2$, i.e., $\mathbf{A} \in \mathbb{C}^{M \times N}$, then $\mathbf{vec}(\mathbf{A})$ is the row major vectorization of a matrix. In this case, we denote the corresponding *Matrix State Preparation* of $\mathbf{A}$ as $\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}} \in \mathbf{MS}_{\alpha}(\mathbf{A})$ [42]. See Section 7.1.1 for related work on state preparation circuit synthesis, including a discussion of resource complexities.

The scale parameter α plays an important role in both block encodings circuits and matrix state preparation circuits. As we saw, it is connected to the probability of successfully preparing desired states (signaled by successfully measuring certain states in the ancillary qubits). Roughly speaking, when α is smaller, shallower circuits suffice or less shots are required in downstream uses. The minimum value is the amount we need to "stretch" or "shrink" a non-unitary matrix $\mathbf{A}$ so it fits within quantum unitary constraints. It is possible to build an exact block encoding only when $\mathbf{A}$ is a scaled unitary matrix. In contrast, for *any* $M \times N$ matrix it is possible to construct a $O(MN)$ depth exact matrix state preparation circuit in $O(MN \log MN)$ (classical) time [51].

2.4 Permutations of qubits

Let S_q denote the set of permutations on $\{0, \ldots, q-1\}$. For a permutation $\sigma \in S_q$, let $\mathcal{S}_{\sigma}$ denote the q-qubit quantum circuit that permutes the qubits according to σ, i.e., qubit i get re-labeled as qubit $\sigma(i)$. Since we talking about permutation on qubits (and not on basis states), $\mathcal{S}_{\sigma}$ can be implemented using SWAP gates for any $\sigma \in S_q$. Note that for the identity permutation id_q we have $\mathcal{S}_{\mathrm{id}_q} = \mathcal{I}_q$ where $\mathcal{I}_q$ is the identity circuit (empty circuit).

Given two permutations $\sigma_1 \in S_q$ and $\sigma_2 \in S_k$, let $\sigma_1 \oplus \sigma_2$ denote the concatenation (aka direct sum) of the permutations:

$$(\sigma_1 \oplus \sigma_2)(j) = \begin{cases} \sigma_1(j) & 0 \leq j < q \\ q + \sigma_2(j - q) & q \leq j < q + k \end{cases}$$

We have $\mathcal{S}_{\sigma_1 \oplus \sigma_2} = \mathcal{S}_{\sigma_1} \otimes \mathcal{S}_{\sigma_2}$. The reason that the direct sum translates to a tensor product is because the permutation is on the qubits, and not the states.

2.5 Describing the complexity of quantum circuits

The algorithms we describe in this paper receive classical inputs and output classical outputs. The output is always a quantum circuit that is to be executed on quantum hardware. Inputs can be classical scalar, vectors and matrices or quantum circuits that describe such objects. Note that the latter are also classical inputs: they are classical descriptions of quantum circuits. When analyzing such algorithms (classical-to-classical algorithms that produce quantum circuits), we need to assess both the classical cost of the algorithm, and the quality of the output circuits. The former is straightforward, while the latter is intricate.

There are quite a few ways to understand the quality of a quantum circuit. One clear metric is the number of qubits used; recall that we denote this by $q(\mathcal{U})$. Two other measures is the overall number of gates used (known as *gate complexity*) and the length of the critical path in the quantum circuits (known as *circuit depth*). We denote the depth and gate complexity of a circuit $\mathcal{U}$ (respectively). We are mostly interested in *surplus complexity*, which are costs (depth, gate complexity, ...) in addition to input circuits.

Both gate complexity and circuit depth depend on a prescribed *gate library*. Typically, quantum hardware allow only for a certain library of elementary gates. Quantum computing frameworks decompose quantum circuits submitted by the user to circuits that use only elementary gates. This a process is called transpilation or compilation. The actual circuit depth and gate complexity of the circuit submitted to hardware (after transpilation) is the real metric of interest. However, to keep our discussion simple, we make use of high level gates in circuits description throughout the paper. Specifically, we allow all single qubit gates, CNOT, fanout-CNOT.

Allowing all single qubit gates gives only a partial view of the overall quality of the quantum circuit. In analysis of quantum circuits, one common way to address this gap is to allow (in the gate library) all Clifford (H, S, CNOT) gates + T gate, and count the complexity in term of T gates (i.e., T-depth and T-count, denoted by $d_T(\mathcal{U})$ and $g_T(\mathcal{U})$). The underlying motivation is that in fault-tolerant architectures, Clifford operations are practically free, so the real cost driver are T gates in the circuit, making T-count and T-depth a good proxy for both space and time complexity of executing a quantum circuit that uses only Clifford+T gates [2, 50]. We also make use of this abstraction, making sure to specify the Clifford+T complexity of single qubit gates we use (see following discussion for key gates used). Again, we mostly interested in surplus complexity compared to the input's T-depth and T-count.

Remark 4. Logically, the depth of the circuit is an upper bound on the T-depth. However, we measure both quantities using a different gate library: regular depth assumes the use of all single qubit gates and CNOTs, while T-depth allows only Clifford+T. Thus, it is possible for the T-depth, as we measure it in this paper, to be larger the the depth (as we measure it in this paper).

T-costs of rotation gates. The T-cost of rotation gates $\mathcal{R}_y(\theta)$ and $\mathcal{R}_z(\theta)$ depends on the angle θ. In general, exact synthesis is only possible for angles of the specific form $\theta = {}^{k\pi}/_{2^n}$ (for integers k, n). For these angels the T-count scales as $O(n)$ [30, 20, 49]. Certain special angles have reduced T-costs: $\theta \in \{0, \pi/2, \pi\}$ are Clifford gates (T-cost = 0), and $\theta = \pi/4$ is a T-gate (T-cost = 1). For arbitrary angles, both $\mathcal{R}_y(\theta)$ and $\mathcal{R}_z(\theta)$ must be approximated using Clifford+T decompositions, with T-costs scaling as $O(\log(1/\epsilon))$ for precision ϵ [16, 48]. For state preparation circuits, the size of the angles depend on the number of prepared amplitudes and this affects the T-depth of the rotations. In general, when preparing N amplitudes, you need $O(\log(N/\epsilon))$ T-gates for each rotation, where ϵ is an accuracy parameter [39]. However, we make the assumption that ϵ is a fixed small number, which allows us to ignore the $\log(1/\epsilon)$ term.

Diagonal Gate. A diagonal gate $\mathcal{D}$ on a m-qubit register is a circuit that applies independent phase factors to each computational basis state, i.e.,

$$\mathbf{M}(\mathcal{D}) = \mathbf{diag}\left(e^{\mathrm{i}\phi_0}, \ldots, e^{\mathrm{i}\phi_{M-1}}\right)$$

for some (input) $\phi_0, \phi_1, \ldots, \phi_{M-1} \in [0, 2\pi)$. Note that while it is typically referred to as a "gate", it is actually a circuit family, and several implementations of it have been proposed in the literature. In our constructions, we write it as a gate, even though we actually mean that any implementation can be used. In complexity analysis, consistent with state-of-the-art results, we assume that gate and depth complexities of a m-qubit diagonal gate are $O(M/m)$, and the T-costs are $d_T(\mathcal{D}) = O(M)$ and $g_T(\mathcal{D}) = O(Mm)$ [54, 39]. We note that structured set of phases, such as multiplexed $\mathcal{R}_z$ rotations for certain rotations, admit significantly lower T-cost.

Remark 5. In reality, a diagonal gate cannot be implemented exactly for arbitrary phases (though there are phases for which it can be implemented exactly), and gate constructions implement the operation up to

a prescribed approximation error ϵ. The T-complexities of state-of-the-art diagonal gate constructions are actually $g_T(\mathcal{D}) = O(M \log(M/\varepsilon))$, $d_T(D) = O(\frac{M}{m} \log(M/\varepsilon))$. However, we again make the assumption that ϵ is a fixed small number, which allows us to ignore the $\log(1/\epsilon)$ term. For simplicity, we generally describe the operation of the circuit as if diagonal gates are implemented exactly.

Eliminate Permutations. In our previous work [42, Appendix A.4], we describe a method, called EliminatePermutations, to eliminate trailing and leading SWAP gates in hypermatrix state preparation circuits. The EliminatePermutations algorithm receives a state preparation circuit $\mathcal{U}_{\mathcal{A}}^{\mathrm{SP}}$ for some hypermatrix $\mathcal{A}$, and assumes the circuit can be written as $\mathcal{U}_{\mathcal{A}}^{\mathrm{SP}} = \mathcal{S}_{\sigma_1} \cdot \mathcal{Q} \cdot \mathcal{S}_{\sigma_2}$ for some (possibly empty) circuit $\mathcal{S}_{\sigma_1}$ and $\mathcal{S}_{\sigma_2}$ that are composed only of SWAP gates, so implement qubit permutations. It outputs a circuit $\mathcal{V}_{\mathcal{A}}^{\mathrm{SP}}$ that also implements a hypermatrix state preparation circuit for $\mathcal{A}$, without leading and trailing SWAPs. Note that typically $\mathbf{M}(\mathcal{U}_{\mathcal{A}}^{\mathrm{SP}}) \neq \mathbf{M}(\mathcal{V}_{\mathcal{A}}^{\mathrm{SP}})$, however, since both circuits implement the same hypermatrix state preparation, in many cases $\mathcal{V}_{\mathcal{A}}^{\mathrm{SP}}$ can replace $\mathcal{U}_{\mathcal{A}}^{\mathrm{SP}}$ in more elaborate circuits that use $\mathcal{U}_{\mathcal{A}}^{\mathrm{SP}}$ as a sub-circuit. The classical cost of the algorithm is $O(g(\mathcal{U}_{\mathcal{A}}^{\mathrm{SP}}))$.

3 Multiplexers and rotation multiplexers (aka uniformly controlled rotations)

Quantum multiplexers (also known as *uniformly controlled circuits*) are central to the various algorithms we propose in this paper. In this section, we discuss both general construction of mulitplexers, and ones that are specific to rotation mulitplexers.

Quantum multiplexers are fundamental building blocks in quantum circuits [9, 57, 43]. They enable *if-then-else* conditional operations on target qubits based on the state of select (control) qubits. In circuit diagrams and figures, quantum multiplexers are typically drawn with two components: an empty rectangular box on the control lines representing all the control states, and a target gate on the target qubits representing the family of circuits. Figure 1 (a) illustrates the graphical notation commonly used to denote multiplexing a parameterized family of circuits $\{\mathcal{U}_\theta\}$. Mathematically, a multiplexer $\mathcal{U}$ is usually described via the equation

$$\mathcal{U} = \sum_{i=0}^{M-1} |i\rangle \langle i| \otimes \mathcal{U}_i$$

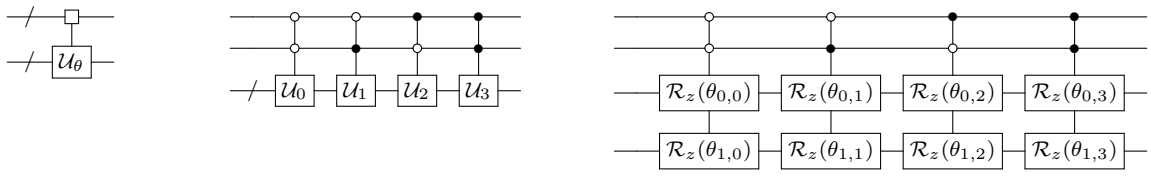
meaning that the operation "selects" which circuit $(\mathcal{U}_0, \ldots, \mathcal{U}_{M-1})$ to apply based on the state of the $m = \log_2 M$ qubits in the control register. Thus, in the literature on Linear Combinations of Unitaries (LCU), the operation is referred to as SELECT operation, and is an important building block in implementing LCUs.

A multiplexer is formally defined as a circuit whose associated unitary matrix is the direct sum of the unitaries corresponding to the multiplexed circuits.

Definition 6 (Direct sum between matrices)**.** Given M matrices $\mathbf{A}_0, \ldots, \mathbf{A}_{M-1}$, their *direct sum*, denoted by $\bigoplus_{i=0}^{M-1} \mathbf{A}_i$, is the block diagonal matrix obtained by putting $\mathbf{A}_0, \ldots, \mathbf{A}_{M-1}$ on the diagonal:

$$\bigoplus_{i=0}^{M-1} \mathbf{A}_i \coloneqq \mathbf{blkdiag}\left(\mathbf{A}_0, \ldots, \mathbf{A}_{M-1}\right) = \begin{bmatrix} \mathbf{A}_0 & & & \\ & \mathbf{A}_1 & & \\ & & \ddots & \\ & & & \mathbf{A}_{M-1} \end{bmatrix}$$

Definition 7 (Multiplexer circuit)**.** Suppose that $\mathcal{U}_0, \ldots, \mathcal{U}_{M-1}$ are circuits that on the same number of qubits. A circuit that implements a *multiplexer between* $\mathcal{U}_0, \ldots, \mathcal{U}_{M-1}$, denoted by $\mathcal{U} \in \mathcal{MX}(\mathcal{U}_0, \mathcal{U}_1, \ldots, \mathcal{U}_{M-1})$,



(a) (b) (c)

Figure 1: Quantum multiplexer constructions. (a) Notation for quantum multiplexer for a family of circuits $\{\mathcal{U}_\theta\}$. (b) Generic construction of multiplexer between $\mathcal{U}_0, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3$. (c) Generic construction of 2D $\mathcal{R}_z$ multiplexer with angles in a matrix $\mathbf{\Theta}$.

is any circuit $\mathcal{U}$ for which

$$\mathbf{M}(\mathcal{U}) = \bigoplus_{i=0}^{M-1} \mathbf{M}(\mathcal{U}_i)$$

Note that $q(\mathcal{U}) = n + q(\mathcal{U}_0)$.

Figure 1 (b) illustrates a generic (and naive) implementation of a quantum multiplexer that works for aribtrary input circuits, but uses, as its building block, multi-qubit control versions of input circuits (the example shown has $M = 4$, and thus utilizes two control qubits).

3.1 Construction of rotation multiplexers

Multi-qubit controls are costly, thus the generic implementation is inefficient. Since in many cases much of the complexity in a quantum circuit comes from a multiplexer subcircuit, more efficient multiplexers have been proposed for certain cases. One such case is multiplexers of rotation gates.

Definition 8. Let M be a power of two, and $K \geq 1$. Given angles $\theta_{0,0}, \ldots, \theta_{(K-1),(M-1)} \in [0, 2\pi)$, a K-by-M $\mathcal{R}_y$ 2D-multiplexer of these angles is any circuit $\mathcal{U}$ such that

$$\begin{aligned}\mathcal{U} \in \mathcal{MX}\big(&\mathcal{R}_y(\theta_{0,0}) \otimes \cdots \otimes \mathcal{R}_y(\theta_{K-1,0}),\\ &\mathcal{R}_y(\theta_{0,1}) \otimes \cdots \otimes \mathcal{R}_y(\theta_{K-1,1}),\\ &\vdots\\ &\mathcal{R}_y(\theta_{0,M-1}) \otimes \cdots \otimes \mathcal{R}_y(\theta_{K-1,M-1})\big)\end{aligned}$$

When $K = 1$ we say this is a 1D multiplexer. We also introduce the notation $\mathcal{U} \in \mathcal{MX}_y(\mathbf{\Theta})$, where $\mathbf{\Theta} \in \mathbb{R}^{K \times M}$ is the matrix with angles given by $\theta_{0,0}, \ldots, \theta_{K-1,M-1}$. The definition of $\mathcal{R}_z$ 1D and 2D multiplexer simply replaces $\mathcal{R}_y$ gates with $\mathcal{R}_z$ gates.

Figure 1 (c) illustrates the construction of a 2-by-4 $\mathcal{R}_z$ 2D multiplexer via its naive implementation. Mottonen et al. showed how to efficiently implement $\mathcal{R}_y$ (respectively $\mathcal{R}_z$) 1D multiplexers via only $\mathcal{R}_y$ (resppectively $\mathcal{R}_z$) gates and CNOT gates [44]. The technique is illustrated in Figure 2. Since then, the construction has been repeated as basic building block in many works [52, 10, 51, 11].

We now give more explicit details on the construction of Mottonen et al. These details are required to for some of our proofs. First we consider the 1D-multiplexer case ($K = 1$). The construction for $\mathcal{R}_y$ and $\mathcal{R}_y$ is essentially the same, with the difference being the gate used at the bottom of the recursion. Thus, in the following we use ξ as a generic placeholder for either y or z.

Given a row vector of angles $\boldsymbol{\theta}$, the goal is to build circuits $1d\mathcal{MUX}\mathcal{R}_\xi(\boldsymbol{\theta}) \in \mathcal{MX}_\xi(\boldsymbol{\theta})$. In $1d\mathcal{MUX}\mathcal{R}_\xi(\boldsymbol{\theta})$ there is one target qubit t and m control qubits labeled $c_0, \ldots, c_{m-1}$. We first compute a new row vector of angles $\hat{\boldsymbol{\theta}} = \frac{1}{\sqrt{M}}\boldsymbol{\theta}\mathbf{H}_M\mathbf{G}_M$ where $\mathbf{H}_M$ is the M-by-M Walsh-Hadamard matrix, and and $\mathbf{G}_M$ is the permutation matrix that reorders from binary order to the binary-reflected Gray code (BRGC) order (i.e., index j moves to index $j \oplus (j \gg 1)$ where $j \gg 1$ denotes shift of bits right). From there, the construction is

$$1d\mathcal{MUX}\mathcal{R}_\xi(\boldsymbol{\theta}) = \mathcal{F}_m^{(0)}(\hat{\boldsymbol{\theta}}) \cdot \mathrm{CX}(c_0, t)$$

where $\mathcal{F}_m^{(0)}(\hat{\boldsymbol{\theta}})$ is specified recursively (with superscript denoting depth of recursion):

$$\mathcal{F}_m^{(d)}(\boldsymbol{\phi}) \coloneqq \mathcal{F}_m^{(d+1)}(\boldsymbol{\phi}_{0:(2^{m-d-1}-1)}) \cdot \mathrm{CX}(c_d, t) \cdot \mathcal{F}_m^{(d+1)}(\boldsymbol{\phi}_{2^{m-d-1}:(2^{m-d}-1)}) \tag{2}$$

The leaf case is:

$$\mathcal{F}_m^{(m-1)}(\alpha, \beta) \coloneqq \mathcal{R}_{\xi,t}(\alpha) \cdot \mathrm{CX}(c_{m-1}, t) \cdot \mathcal{R}_{\xi,t}(\beta) \tag{3}$$

In the above, $\mathrm{CX}(c, t)$ denotes a CNOT gate with control c and target t, and $\mathcal{R}_{\xi,t}(\gamma)$ denotes adding a single $\mathcal{R}_\xi$ rotation by angle γ on target t. For simplicity, we allow omitting the rotation label when clear from context, since the circuit construction is identical for both $\mathcal{R}_z$ and $\mathcal{R}_y$ multiplexers, differing only in the leaf rotations.

The technique can be easily extended to 2D rotation multiplexers, repeating the 1D multiplexer on "rows" of the circuit, using the same control qubits. The CNOTs become fanout-CNOTS[3]. See Figure 3 for an illustration. In the following, we denote this construction by $2d\mathcal{MUX}\mathcal{R}_y(\boldsymbol{\Theta})$ and by $2d\mathcal{MUX}\mathcal{R}_z(\boldsymbol{\Theta})$ where $\boldsymbol{\Theta} \in \mathbb{R}^{K \times M}$ is a matrix of rotation angles. A key observation is that when implementing $2d\mathcal{MUX}\mathcal{R}_y(\boldsymbol{\Theta})$ (respectively $2d\mathcal{MUX}\mathcal{R}_z(\boldsymbol{\Theta})$) we set up an array of $\mathcal{R}_y$ (respectively, $\mathcal{R}_z$) gates with the same layout as $\boldsymbol{\Theta}$, with CNOT gates in between. The following fact relates the rotation angles to the angles in the gates of the multiplexers.

Fact 9 ([43])**.** *The angles in the rotation gates of $1d\mathcal{MUX}\mathcal{R}_z(\boldsymbol{\theta})$ and $1d\mathcal{MUX}\mathcal{R}_y(\boldsymbol{\theta})$ are given by $\hat{\boldsymbol{\Theta}} = \frac{1}{\sqrt{M}}\boldsymbol{\Theta}\mathbf{H}_M\mathbf{G}_M$ where $\mathbf{H}_M$ is the M-by-M Walsh-Hadamard matrix, and and $\mathbf{G}_M$ is the permutation matrix that reorders from binary order to the binary-reflected Gray code (BRGC) order (i.e., index j moves to index $j \oplus (j \gg 1)$ where $j \gg 1$ denotes shift of bits right).*

The classical cost of computing the $\hat{\boldsymbol{\Theta}}$ using (fast) Welsh-Hadamard transform is $O(KMm)$. The resulting circuit has KM rotations angles translated to gates and M fanout-CNOTs gate (between each columns). The depth, when counting fanout-CNOTs and single control rotation gates as depth 1, is $2M$. For T + Clifford analysis, the T-gate count is $O(KM \log(M/\epsilon))$ (each of the KM arbitrary rotations requires $O(\log(M/\epsilon))$ T-gates for approximation, where ϵ represents the total approximation error when converting arbitrary rotation angles into discrete T-gates) and the T-depth is $O(M \log(M/\epsilon))$ (the overall $2M$ depth is multiplied by the $O(\log(M/\epsilon))$ T-depth of each approximated rotation layer) for an approximation error ϵ of the arbitrary angles.

The discussion above assumed that the angles $\hat{\boldsymbol{\Theta}}$ that form the array of rotation gates is dense and arbitrary. However, it might be the case that $\hat{\boldsymbol{\Theta}}$ is sparse even if $\boldsymbol{\Theta}$ is dense, and vice versa. The quantum complexity of the resulting multiplexer circuit depends on the density of $\hat{\boldsymbol{\Theta}}$, though the classical cost of forming $\hat{\boldsymbol{\Theta}}$ is unchanged (so the classical cost of creating the multiplexer remains the same). In Section 5, we discuss a case in which $\hat{\boldsymbol{\Theta}}$ is not only highly sparse, but also structured with unique entries. We show that in that case, we can build an ad-hoc construction for this multiplexer that is highly efficient, reducing both quantum and classical costs.

[3] A sequence of CNOT gates where the same control qubit is used to target multiple different qubits (sequentially with depth 1).

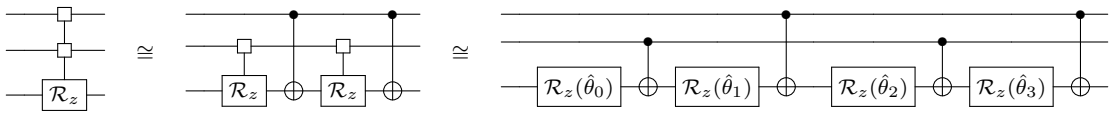


Figure 2: Illustration of the recursive construction of [44] for 1D $\mathcal{R}_z$-rotation multiplexer of angles $\theta_0, \theta_1, \theta_2, \theta_3$. The angles $\hat{\theta}_0, \hat{\theta}_1, \hat{\theta}_2, \hat{\theta}_3$ are computed as linear combination of $\theta_0, \theta_1, \theta_2, \theta_3$.

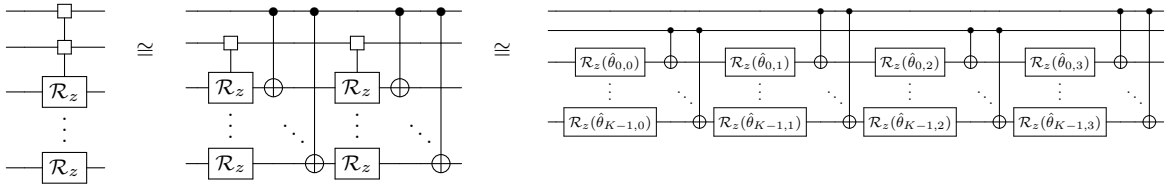


Figure 3: Illustration of the recursive construction of 2D $\mathcal{R}_z$rotation multiplexer of angles $\mathbf{\Theta} \in \mathbb{R}^{K \times 4}$. The angles $\hat{\mathbf{\Theta}}$ are computed by $\hat{\mathbf{\Theta}} = \frac{1}{2}\mathbf{\Theta}\mathbf{H}_4\mathbf{G}_4$.

4 Converting between standard and Pauli basis

The Pauli decomposition of matrices plays a fundamental role in quantum computing and quantum information theory, providing an efficient means of representing and manipulating quantum operators. We make extensive use of this decomposition in our constructions. In this section, we discuss how to convert between the standard and Pauli basis representations of a matrix, both classically and quantumly. While the classical case has been addressed in the literature, the quantum case has not.

4.1 Pauli matrices

The Pauli matrices, along with the identity matrix, form a orthogonal basis for 2×2 matrices.

Definition 10 (Pauli matrices)**.** The Pauli matrices are:

$$\sigma_0 \coloneqq \sigma_I = \mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$\sigma_1 \coloneqq \sigma_X = \mathbf{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$
$$\sigma_2 \coloneqq \sigma_Y = \mathbf{Y} = \begin{bmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{bmatrix}$$
$$\sigma_3 \coloneqq \sigma_Z = \mathbf{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Fact 11. *The following hold:*

- ***Orthogonality with respect to*** $\langle \cdot, \cdot \rangle_{\mathbb{H}_2}$***:*** $\mathbf{Tr}(\sigma_i \sigma_j) = 2\delta_{ij}$ *where* δ_{ij} *is the Kronecker delta.*
- ***Anti-commutation:*** $\sigma_i \sigma_j = -\sigma_j \sigma_i$ *for* $i \neq j$, $i, j \in \{1, 2, 3\}$.
- ***Multiplication:*** $\sigma_i^2 = \mathbf{I}$ *for all* i.
- ***Basis Completeness:*** *Any* 2×2 *Hermitian matrix* $\mathbf{A}$ *can be uniquely expressed as* $\mathbf{A} = \sum_{i=0}^{3} \alpha_i \sigma_i$*, where* $\alpha_i \in \mathbb{R}$*, and any* 2×2 *complex matrix* $\mathbf{B}$ *can be uniquely expressed as* $\mathbf{B} = \sum_{i=0}^{3} \beta_i \sigma_i$ *where* $\beta_i \in \mathbb{C}$.

Going to higher order, we have the following.

Definition 12 (Higher-order Pauli matrices)**.** Let $\Sigma = \{I, X, Y, Z\}$ and $N = 2^n$. For a *word* (i.e., sequence) $\mathbf{w} = (w_0, w_2, \ldots, w_{n-1}) \in \Sigma^n$, the *n-wise Pauli matrix* corresponding to $\mathbf{w}$ is defined as:

$$\sigma_{\mathbf{w}} \coloneqq \sigma_{w_0} \otimes \sigma_{w_1} \otimes \cdots \otimes \sigma_{w_{n-1}} \in \mathbb{H}_N$$

In quantum computing and quantum information, such matrices are typically referred to *multi-qubit Pauli matrices*. However, their description is completely classical, so we opted not to use the confusing term "qubit". It is also more notation efficient and more convenient to write $\mathbf{w}$ as string $\mathbf{w} = w_0 w_1 \cdots w_{n-1}$, and use juxtaposition to denote word concatenation, i.e., $\mathbf{w}_1\mathbf{w}_2 \coloneqq w_{11} \cdots w_{1,k} w_{21} \cdots w_{2,m}$. In term of indices, it is useful to identify the symbols I, X, Y, Z as $I = 0$, $X = 1$, $Y = 2$, $Z = 3$, so, for example, we view the word $XYZIX$ as synonymous to $(1, 2, 3, 0, 1)$.

Assume $N = 2^n$. Consider the set $\mathbb{P}_N \coloneqq \{\sigma_{\mathbf{w}} : \mathbf{w} \in \Sigma^n\} \subseteq \mathbb{H}_N$. Clearly, there are $|\Sigma|^n = N^2$ such matrices. Since they are the Kronecker product of Hermitian matrices, they are also Hermitian. Due to the mixed-product property of the Kronecker, and the fact that trace of Kronecker product is the product of traces, for each $\mathbf{w}, \mathbf{s} \in \Sigma^q$ we have

$$\langle \sigma_{\mathbf{w}}, \sigma_s \rangle_{\mathbb{H}_N} = \begin{cases} N & \mathbf{w} = \mathbf{s} \\ 0 & \mathbf{w} \neq \mathbf{s} \end{cases}$$

Recall that the dimension of $\mathbb{H}_N$ as a real vector space is N^2, so the set $\mathbb{P}_N$ forms an orthogonal basis for $\mathbb{H}_N$ over reals. They are also an orthogonal basis for $\mathbb{C}^{N\times N}$ over $\mathbb{C}$ with respect to the Hilbert-Schmidt inner product.

4.2 Pauli decomposition and its (classical) computation

Higher-order Pauli matrices form an orthogonal basis of $\mathbb{C}^{N\times N}$ over $\mathbb{C}$, so every matrix $\mathbf{A} \in \mathbb{C}^{N\times N}$ can be uniquely expressed as a linear combination of higher-order Pauli matrices:

$$\mathbf{A} = \sum_{\mathbf{w}\in\Sigma^n} \alpha_{\mathbf{A}}(\mathbf{w}) \sigma_{\mathbf{w}}$$

where $\{\alpha_{\mathbf{A}}(\mathbf{w})\}_{\mathbf{w}\in\Sigma^n} \subseteq \mathbb{C}$ are the N^2 *Pauli coefficients* of $\mathbf{A}$. When $\mathbf{A}$ is Hermitian, we have $\alpha_{\mathbf{A}}(\mathbf{w}) \in \mathbb{R}$ for all $\mathbf{w} \in \Sigma^n$. For every $\mathbf{w}$, view $\mathbf{A} \mapsto \alpha_{\mathbf{A}}(\mathbf{w})$ as function from $\mathbb{C}^{N\times N}$ (respectively $\mathbb{H}_N$) to $\mathbb{C}$ (respectively $\mathbb{R}$). Alternatively, for any matrix $\mathbf{A}$ we can view $\mathbf{w} \mapsto \alpha_{\mathbf{A}}(\mathbf{w})$ as a function from Σ^n to $\mathbb{C}$ (or $\mathbb{R}$ for Hermitian $\mathbf{A}$). Since we can identify each word with an index between 0 and $N^2 - 1$ (using lexigraphical ordering), we can view $\alpha_{\mathbf{A}}$ as a vector of length N^2.

We can also arrange the Pauli coefficients in the form of a hypermatrix, using the identification $I = 0$, $X = 1$, $Y = 2$, $Z = 3$. We denote this hypermatrix by $\mathcal{A}_P$. When $\mathbf{A}$ is $N \times N$, the order of $\mathcal{A}_P$ is n, and its dimension is $4 \times 4 \times \cdots \times 4$. Since the higher order Pauli matrices form an orthogonal basis, and the norm of each basis matrix is $\sqrt{N}$, we have $\|\mathbf{A}\|_F^2 = N\|\mathcal{A}_P\|_F^2$.

Example 13. Let's consider a simple case of a 4×4 Hermitian matrix. Its Pauli decomposition involves 16 terms. The hypermatrix, here a matrix, $\mathbf{A}_P$ is 4×4, where each element corresponds to a specific Pauli coefficient:

$$\mathbf{A}_P = \begin{bmatrix} \alpha_{II} & \alpha_{IX} & \alpha_{IY} & \alpha_{IZ} \\ \alpha_{XI} & \alpha_{XX} & \alpha_{XY} & \alpha_{XZ} \\ \alpha_{YI} & \alpha_{YX} & \alpha_{YY} & \alpha_{YZ} \\ \alpha_{ZI} & \alpha_{ZX} & \alpha_{ZY} & \alpha_{ZZ} \end{bmatrix}$$

Using the orthogonality of $\mathbb{P}_N$, a simple formula for the Pauli coefficients can be obtained:

$$\alpha_{\mathbf{A}}(\mathbf{w}) = \frac{\mathbf{Tr}\left(\sigma_{\mathbf{w}}\mathbf{A}\right)}{N}.$$

Computing the Pauli coefficients via direct evaluation of this formula is computationally expensive, with total cost of $O(N^5)$. However, by exploiting the structure of $\sigma_{\mathbf{w}}$ and the properties of Pauli matrices, we can derive a substantially more efficient evaluation method and reduce the cost to $O(N^2 n)$. The basic idea is to use *Pauli submatrices* and *Pauli subcoefficents*.

Definition 14 (Pauli submatrices)**.** Given $\mathbf{A} \in \mathbb{C}^{N\times N}$, and for each $c \in \Sigma$, we define the Pauli submatrix $\mathbf{A}_c \in \mathbb{C}^{(N/2)\times(N/2)}$ as:

$$\mathbf{A}_c \coloneqq \sum_{\mathbf{w}\in\Sigma^{n-1}} \alpha_{\mathbf{A}}(c\mathbf{w})\sigma_{\mathbf{w}}$$

When $\mathbf{A}$ is Hermitian, $\mathbf{A}_c$ is also Hermitian.

Example 15. Given $\mathbf{A}$, we have

$$\begin{aligned}
\mathbf{A}_I &= \sum_{\mathbf{w}\in\Sigma^{n-1}} \alpha_{\mathbf{A}}(I\mathbf{w})\sigma_{\mathbf{w}} \\
\mathbf{A}_X &= \sum_{\mathbf{w}\in\Sigma^{n-1}} \alpha_{\mathbf{A}}(X\mathbf{w})\sigma_{\mathbf{w}} \\
\mathbf{A}_Y &= \sum_{\mathbf{w}\in\Sigma^{n-1}} \alpha_{\mathbf{A}}(Y\mathbf{w})\sigma_{\mathbf{w}} \\
\mathbf{A}_Z &= \sum_{\mathbf{w}\in\Sigma^{n-1}} \alpha_{\mathbf{A}}(Z\mathbf{w})\sigma_{\mathbf{w}}
\end{aligned}$$

Definition 16 (Pauli subcoefficient)**.** Given $\mathbf{A} \in \mathbb{C}^{N\times N}$, for each $c \in \Sigma$ define the mapping $\alpha_{\mathbf{A},c} : \Sigma^{n-1} \to \mathbb{C}$ as follows:

$$\alpha_{\mathbf{A},c}(\mathbf{w}) \coloneqq \alpha_{\mathbf{A}}(c\mathbf{w})$$

The result is real if $\mathbf{A}$ is Hermitian.

Lemma 17. Any matrix $\mathbf{A} \in \mathbb{C}^{N\times N}$ can be expressed as:

$$\mathbf{A} = \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{A}_I + \mathbf{A}_Z & \mathbf{A}_X - \mathrm{i}\mathbf{A}_Y \\ \mathbf{A}_X + \mathrm{i}\mathbf{A}_Y & \mathbf{A}_I - \mathbf{A}_Z \end{bmatrix} \tag{4}$$

Proof. Due to the bilinearity and associativity of the Kronecker product we have

$$\begin{aligned}
\mathbf{A} &= \sum_{\mathbf{w}\in\Sigma^n} \alpha_{\mathbf{w}}\sigma_{\mathbf{w}} \\
&= \sum_{\mathbf{w}\in\Sigma^{n-1}} \alpha_{\mathbf{A},I}(\mathbf{w})\sigma_{I\mathbf{w}} + \sum_{\mathbf{w}\in\Sigma^{n-1}} \alpha_{\mathbf{A},X}(\mathbf{w})\sigma_{X\mathbf{w}} + \sum_{\mathbf{w}\in\Sigma^{n-1}} \alpha_{\mathbf{A},Y}(\mathbf{w})\sigma_{Y\mathbf{w}} + \sum_{\mathbf{w}\in\Sigma^{n-1}} \alpha_{\mathbf{A},Z}(\mathbf{w})\sigma_{Z\mathbf{w}} \\
&= \sum_{\mathbf{w}\in\Sigma^{n-1}} \alpha_{\mathbf{A},I}(\mathbf{w})(\sigma_I \otimes \sigma_{\mathbf{w}}) + \sum_{\mathbf{w}\in\Sigma^{n-1}} \alpha_{\mathbf{A},X}(\mathbf{w})(\sigma_X \otimes \sigma_{\mathbf{w}}) + \sum_{\mathbf{w}\in\Sigma^{n-1}} \alpha_{\mathbf{A},Y}(\mathbf{w})(\sigma_Y \otimes \sigma_{\mathbf{w}}) + \sum_{\mathbf{w}\in\Sigma^{n-1}} \alpha_{\mathbf{A},Z}(\mathbf{w})(\sigma_Z \otimes \sigma_{\mathbf{w}}) \\
&= \sigma_I \otimes \left(\sum_{\mathcal{W}\in\Sigma^{n-1}} \alpha_{\mathbf{A},I}(\mathbf{w})\sigma_{\mathbf{w}} \right) + \sigma_X \otimes \left(\sum_{\mathbf{w}\in\Sigma^{n-1}} \alpha_{\mathbf{A},X}(\mathbf{w})\sigma_{\mathbf{w}} \right) + \sigma_Y \otimes \left(\sum_{\mathbf{w}\in\Sigma^{n-1}} \alpha_{\mathbf{A},Y}(\mathbf{w})\sigma_{\mathbf{w}} \right) + \sigma_Z \otimes \left(\sum_{\mathbf{w}\in\Sigma^{n-1}} \alpha_{\mathbf{A},Z}(\mathbf{w})\sigma_{\mathbf{w}} \right) \\
&= \sigma_I \otimes \mathbf{A}_I + \sigma_X \otimes \mathbf{A}_X + \sigma_Y \otimes \mathbf{A}_Y + \sigma_Z \otimes \mathbf{A}_Z \\
&= \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{A}_I + \mathbf{A}_Z & \mathbf{A}_X - i\mathbf{A}_Y \\ \mathbf{A}_X + i\mathbf{A}_Y & \mathbf{A}_I - \mathbf{A}_Z \end{bmatrix}
\end{aligned}$$

□

Given $\mathbf{A} \in \mathbb{H}_N$, we can write $\mathbf{A}$ in block form as:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{12}^{\mathrm{H}} & \mathbf{A}_{22} \end{bmatrix}$$

where $\mathbf{A}_{11}, \mathbf{A}_{22} \in \mathbb{H}_{N/2}$ and $\mathbf{A}_{12} \in \mathbb{C}^{(N/2)\times(N/2)}$. Then, the Pauli submatrices of $\mathbf{A}$ are given by:

$$\begin{aligned}
\mathbf{A}_I &= \frac{1}{\sqrt{2}}(\mathbf{A}_{11} + \mathbf{A}_{22}) \\
\mathbf{A}_X &= \frac{1}{\sqrt{2}}(\mathbf{A}_{12} + \mathbf{A}_{12}^{\mathrm{H}}) \\
\mathbf{A}_Y &= \frac{\mathrm{i}}{\sqrt{2}}(\mathbf{A}_{12} - \mathbf{A}_{12}^{\mathrm{H}}) \\
\mathbf{A}_Z &= \frac{1}{\sqrt{2}}(\mathbf{A}_{11} - \mathbf{A}_{22})
\end{aligned}$$

Thus, for any word $\mathbf{w} \in \Sigma^{n-1}$ we have:

$$\begin{aligned}
\alpha_{\mathbf{A},I}(\mathbf{w}) &= \frac{1}{\sqrt{2}}(\alpha_{\mathbf{A}_{11}}(\mathbf{w}) + \alpha_{\mathbf{A}_{22}}(\mathbf{w})) \\
\alpha_{\mathbf{A},X}(\mathbf{w}) &= \frac{\alpha_{(\mathbf{A}_{12}+\mathbf{A}_{12}^{\mathrm{H}})}(\mathbf{w})}{\sqrt{2}} \\
\alpha_{\mathbf{A},Y}(\mathbf{w}) &= \frac{\alpha_{(\mathrm{i}(\mathbf{A}_{12}-\mathbf{A}_{12}^{\mathrm{H}}))}(\mathbf{w})}{\sqrt{2}} \\
\alpha_{\mathbf{A},Z}(\mathbf{w}) &= \frac{1}{\sqrt{2}}(\alpha_{\mathbf{A}_{11}}(\mathbf{w}) - \alpha_{\mathbf{A}_{22}}(\mathbf{w}))
\end{aligned}$$

Using the above observation, we can develop an efficient recursive algorithm for computing the Pauli decomposition of a Hermitian matrix. First, we compute the Pauli decompositions of four $N/2$-by-$N/2$ matrices: $\mathbf{A}_{11}, \mathbf{A}_{22}, \mathbf{A}_{12}+\mathbf{A}_{12}^{\mathrm{H}}, \mathrm{i}(\mathbf{A}_{12}-\mathbf{A}_{12}^{\mathrm{H}})$. We then combine the resulting coefficeints to compute the Pauli decomposition of $\mathbf{A}$. The bottom of the recursion, which amounts to computing the Pauli decomposition of a 2-by-2 matrix, is trivial. The procedure is summarized in Algorithm 1, which returns as output the coefficient tensor $\mathcal{A}_P$. The combination costs $O(N^2)$, so the running time obeys $T(N) = 4T(N/2) + O(N^2)$ which implies the cost of $O(N^2 n)$.

To extend the method to non-Hermitian matrices, we decompose them into two Hermitian parts ($\mathbf{A} = \frac{\mathbf{A}^*+\mathbf{A}}{2} - \mathrm{i}\frac{\mathbf{A}^*-\mathbf{A}}{2i}$), each independently decomposable into real Pauli coefficients. Combining the coefficients, we obtain a Pauli decomposition for the original matrix $\mathbf{A}$.

Related work on efficient computation of Pauli decompositions. Direct computation of the Pauli coefficients for an $N \times N$ matrix using the naive approach costs $O(N^5)$, making it impractical for large matrices. To address this computational challenge, various approaches have been proposed in the literature [47, 29, 31, 58, 34, 24, 26, 19, 25]. Koska et al. introduced a tree-based approach that reduces asymptotic complexity to $O(N^3)$ by hierarchically pruning redundant paths in the Pauli operator tree [31], making it particularly efficient for structured matrices (e.g., tridiagonal systems). Hantzko et al. employed recursive matrix partitioning to avoid costly multiplications with asymptotic scaling of $O(N^2 n)$ for general matrices [26]. Their tensorized Pauli decomposition algorithm is very similar to the one we describe in this section, and was developed in parallel.

Georges et al. presented a method for computing the Pauli coefficients using the Walsh-Hadamard Transform [18] (however, elements of their approach are traceable to a Stack Overflow comment by Gidney [19]). Their method achieves $O(N^2 n)$ complexity, and enables in-place computation with $O(1)$ additional memory requirements. However, their method is only described for classical computing. However, it can be leveraged for quantum conversion, as we discuss in the next subsection.

Algorithm 1 DENSE2PAULI: Computing $\mathcal{A}_P$ for a dense Hermitian matrix $\mathbf{A}$.

Input: $\mathbf{A} \in \mathbb{H}_N$, where N is a power of 2

$n \leftarrow \log_2 N$
if $q == 1$ **then**
  $\mathbf{A}_P \leftarrow \frac{1}{2}\begin{bmatrix} a_{11}+a_{22} & 2\mathrm{Re}(a_{12}) \\ -2\mathrm{Im}(a_{12}) & a_{11}-a_{22} \end{bmatrix}$
else
  Split $\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{12}^* & \mathbf{A}_{22} \end{bmatrix}$ where each matrix is $(N/2) \times (N/2)$
  $\mathcal{P}_{11} \leftarrow$ DENSE2PAULI($\mathbf{A}_{11}$)
  $\mathcal{P}_{22} \leftarrow$ DENSE2PAULI($\mathbf{A}_{22}$)

  $\mathcal{P}_I \leftarrow \mathcal{P}_{11} + \mathcal{P}_{22}$
  $\mathcal{P}_X \leftarrow$ DENSE2PAULI($\mathbf{A}_{12} + \mathbf{A}_{12}^{\mathrm{H}}$)
  $\mathcal{P}_Y \leftarrow$ DENSE2PAULI($i(\mathbf{A}_{12} - \mathbf{A}_{12}^{\mathrm{H}})$)
  $\mathcal{P}_Z \leftarrow \mathcal{P}_{11} - \mathcal{P}_{22}$

  Initialize $\mathcal{A}_P$ to a order n tensor of size $4 \times 4 \times \cdots \times 4$
  $\mathcal{A}_P(0, :, \ldots, :) \leftarrow \frac{1}{2}\mathcal{P}_I$
  $\mathcal{A}_P(1, :, \ldots, :) \leftarrow \frac{1}{2}\mathcal{P}_X$
  $\mathcal{A}_P(2, :, \ldots, :) \leftarrow \frac{1}{2}\mathcal{P}_Y$
  $\mathcal{A}_P(3, :, \ldots, :) \leftarrow \frac{1}{2}\mathcal{P}_Z$
end if

return $\mathcal{A}_P$

4.3 Quantum conversion - from matrix state preparation to Pauli state preparation (and back)

In this subsection, we discuss a algorithm for converting between a description of the matrix in the standard basis (given in the form of a matrix state preparation circuit $\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}}$), to a description in the Pauli basis (in the form of a hypermatrix state preparation circuit $\mathcal{U}_{\mathcal{A}_P}^{\mathrm{SP}}$). Our goal is twofold. First, to show we can do this conversion very efficiently. Second, we need this operation as a component in later algorithms. Our algorithm is based on the Walsh-Hadamard Transform based algorithm from [18], though that paper describes only a classical algorithm. Formally, we state the problem as follows:

Problem 18. Given a matrix state preparation circuit $\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}} \in \mathbf{MS}_{\alpha}(\mathbf{A})$, where $\mathbf{A} \in \mathbb{C}^{N \times N}$, compute hypermatrix state preparation circuit $\mathcal{U}_{\mathcal{A}_p}^{\mathrm{SP}} \in \mathbf{MS}_{f(\alpha,N)}(\mathcal{A}_P)$, where $\mathcal{A}_P$ is the Pauli coefficient hypermatrix of $\mathbf{A}$ and $f(\alpha, N)$ is some function.

Theorem 25 describes an algorithm for solving Problem 18. To state and prove the theorem, we need to define some ancillary circuits, and prove auxiliary results.

Proposition 19. *Let $m \geq 2$ be an integer. Let $\sigma^{(m)}$ be the following permutation on $(0, \ldots, 2m-1)$:*

$$\sigma^{(m)}(j) = \begin{cases} j & \text{if } 0 \leq j < m-1 \text{ or } j = 2m-1 \\ j+1 & \text{if } m-1 \leq j < 2m-2 \\ m-1 & \text{if } j = 2m-2 \end{cases}$$

Denote $\mathcal{S}_{(m)} \coloneqq \mathcal{S}_{\sigma^{(m)}}$. This circuit moves the qubit at position $2m-2$ to position $m-1$, shifting the intermediate qubits forward by one position.

Then,

$$\mathbf{M}(\mathcal{S}_{(m)}) = \mathbf{I}_{2^{m-1}} \otimes \mathbf{P}_{2,2^{m-1}} \otimes \mathbf{I}_2 \tag{5}$$

where $\mathbf{P}_{K,L}$ is the Kronecker permutation matrix, defined as $\mathbf{P}_{K,L} \coloneqq \sum_{i=0}^{K-1}\sum_{j=0}^{L-1} \mathbf{E}_{ij}^{(K\times L)} \otimes \mathbf{E}_{ji}^{(L\times K)}$.

Proof. We prove Eq. (5) by analyzing the action on computational basis states, which gives the columns of $\mathbf{M}(\mathcal{S}_{(m)})$ (the action of $\mathcal{S}_{(m)}$ on $|j\rangle_{2m}$ gives column j in $\mathbf{M}(\mathcal{S}_{(m)})$). Any index $j \in \{0,1,\ldots,2^{2m-1}-1\}$ can be written in binary as $j = \sum_{\ell=0}^{2m-1} b_\ell \cdot 2^\ell$, where $b_\ell \in \{0,1\}$ are the binary digits. Similar to many physics textbooks, we assume that the MSB of i is in the lowest index in the binary expansion, i.e., $|j\rangle_{2m} = |b_0\rangle_1 |b_1\rangle_1 \cdots |b_{2m-1}\rangle_1 =: |b_0 \cdots b_{2m-1}\rangle_{2m}$ where $b_0 \cdots b_{2m-1}$ is the binary expansion of j.[4]

We group the bits as:

$$\begin{aligned}|j\rangle_{2m} &= |b_0,\ldots,b_{m-2}\rangle_{m-1} \otimes |b_{m-1},\ldots,b_{2m-3}\rangle_{m-1} \otimes |b_{2m-2}\rangle_1 \otimes |b_{2m-1}\rangle_1 \\ &= |\text{upper}\rangle_{m-1} \otimes |\text{middle}\rangle_{m-1} \otimes |b_{2m-2}\rangle_1 \otimes |b_{2m-1}\rangle_1\end{aligned}$$

By definition of permutation qubits in the lemma statement, when $\mathcal{S}_{(m)}$ operate on $|j\rangle_{2m}$ we transform

$$|\text{upper}\rangle_{m-1} \otimes |\text{middle}\rangle_{m-1} \otimes |b_{2m-2}\rangle_1 \otimes |b_{2m-1}\rangle_1 \mapsto |\text{upper}\rangle_{m-1} \otimes |b_{2m-2}\rangle_1 \otimes |\text{middle}\rangle_{m-1} \otimes |b_{2m-1}\rangle_1$$

and this defines the jth column of $\mathbf{M}(\mathcal{S}_{(m)})$.

Now, think of a circuit whose matrix is $\mathbf{I}_{2^{m-1}} \otimes \mathbf{P}_{2,2^{m-1}} \otimes \mathbf{I}_2$ which acts on $|j\rangle_{2m}$ (thus giving column j of $\mathbf{I}_{2^{m-1}} \otimes \mathbf{P}_{2,2^{m-1}} \otimes \mathbf{I}_2$). The operation is

$$\begin{aligned}(\mathbf{I}_{2^{m-1}} \otimes \mathbf{P}_{2,2^{m-1}} \otimes \mathbf{I}_2)\,|j\rangle_{2m} &= (\mathbf{I}_{2^{m-1}} \otimes \mathbf{P}_{2,2^{m-1}} \otimes \mathbf{I}_2)(|\text{upper}\rangle_{m-1} \otimes |\text{middle}\rangle_{m-1} \otimes |b_{2m-2}\rangle_1 \otimes |b_{2m-1}\rangle_1) \\ &= |\text{upper}\rangle_{m-1} \otimes \mathbf{P}_{2,2^{m-1}}(|\text{middle}\rangle_{m-1} \otimes |b_{2m-2}\rangle_1) \otimes |b_{2m-1}\rangle_1 \\ &= |\text{upper}\rangle_{m-1} \otimes |b_{2m-2}\rangle_1 \otimes |\text{middle}\rangle_{m-1} \otimes |b_{2m-1}\rangle_1\end{aligned}$$

where we use the fact that $\mathbf{P}_{2,2^{m-1}} = \sum_{i=0}^{1}\sum_{j=0}^{2^{m-1}-1} \mathbf{E}_{ij}^{(2\times 2^{m-1})} \otimes \mathbf{E}_{ji}^{(2^{m-1}\times 2)}$ so for any $|i\rangle_{m-1}|j\rangle_1$ we have $\mathbf{P}_{2,2^{m-1}}|i\rangle_{m-1}|j\rangle_1 = |j\rangle_1|i\rangle_{m-1}$.

We see that the two results are the same. All columns are the same, so the matrices are equal. □

Definition 20 (CNOT Comb Circuit)**.** The *CNOT comb circuit*, denoted as $\mathcal{U}_{\text{comb}}^{(q)}$, is a quantum circuit that acts on $2q$ qubits. The circuit consists of q parallel CNOT gates, where each control qubit $i \in [q-1]$ is connected to its corresponding target qubit $i+q$. In other words, the circuit applies CNOT gates between pairs $(0,q),(1,q+1),\ldots(q-1,2q-1)$. A circuit diagram is shown in Figure 4.

Proposition 21. *The $2q$-qubit CNOT comb circuit $\mathcal{U}_{comb}^{(q)}$ implements a controlled multiplex operation, which can be expressed as:*

$$\mathcal{U}_{comb}^{(q)} \in \mathcal{MX}(\mathcal{U}_0,\ldots,\mathcal{U}_{2^q-1})$$

where for each $k = \sum_{j=0}^{q-1} b(k)_j 2^j$, $\mathbf{M}(\mathcal{U}_k)$ is a tensor product of qPauli X gates based on the binary expansion of k:

$$\mathbf{M}(\mathcal{U}_k) = \bigotimes_{j=0}^{q-1} \mathbf{X}^{b(k)_j}$$

[4]We note that in QISKIT the MSB is the highest index, i.e., $|j\rangle_{2m} = |b_{2m-1}\cdots b_0\rangle_{2m}$.

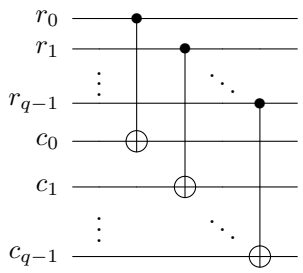


Figure 4: The CNOT Comb circuit, $\mathcal{U}^{(q)}_{\text{comb}}$, on $2q$ qubits. The row qubits (r_i) act as controls on their corresponding target column qubits (c_i).

Proof. We prove by induction on q. For $q = 1$**.** The CNOT comb circuit is a single CNOT gate:

$$\mathbf{M}(\mathcal{U}^{(1)}_{\text{comb}}) = \mathbf{M}(\text{CX}(0,1)) = (|0\rangle\langle 0|)_0 \otimes \mathbf{I} + (|1\rangle\langle 1|)_0 \otimes \mathbf{X}$$

For the inductive step we note that $\mathcal{U}^{(q+1)}_{\text{comb}}$ circuit can be constructed recursively as:

$$\mathcal{U}^{(q+1)}_{\text{comb}} = \mathcal{S}_{(q+1)} \cdot \left(\mathcal{U}^{(q)}_{\text{comb}} \otimes \text{CX}(2q, 2q+1)\right) \cdot \mathcal{S}^{-1}_{(q+1)}$$

where $\mathcal{S}_{(q)}$ defined in Proposition 19 (where we set $m \leftarrow q$). Note that $\mathcal{S}_{(q)} = \mathcal{S}_{(q,q+1)} \cdot \mathcal{S}_{(q+1,q+2)} \cdots \cdots \mathcal{S}_{(2q-1,2q)}$ where the notation (a, b) denotes transposition of a and b.

We assume the proposition holds for q and show for $q \to q+1$. By the inductive hypothesis:

$$\mathbf{M}(\mathcal{U}^{(q)}_{\text{comb}}) = \sum_{k=0}^{2^q-1} |k\rangle\langle k| \otimes \mathbf{M}(\mathcal{U}_k)$$

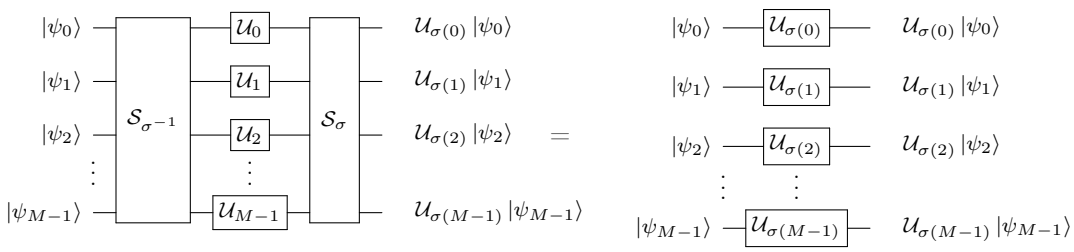


Figure 5: Permutation conjugation property for tensor products of single-qubit operators: $\mathcal{S}_{\sigma^{-1}} \cdot \left(\bigotimes_{i=0}^{M-1} \mathcal{U}_i\right) \cdot \mathcal{S}_\sigma = \bigotimes_{i=0}^{M-1} \mathcal{U}_{\sigma(i)}$. The permutation redistributes operators according to the mapping σ.

and we have

$$\begin{aligned}
\mathbf{M}(\mathcal{U}_{\text{comb}}^{(q+1)}) &= \mathbf{M}(\mathcal{S}_{(q+1)}) \left(\mathbf{M}(\mathcal{U}_{\text{comb}}^{(q)}) \otimes \text{CX}(2q, 2q+1)\right) \mathbf{M}(\mathcal{S}_{(q+1)}^{-1}) \\
&= \mathbf{M}(\mathcal{S}_{(q+1)}) \left(\sum_{k=0}^{2^q-1} |k\rangle\langle k| \otimes \mathbf{M}(\mathcal{U}_k) \otimes [(|0\rangle\langle 0|)_{2q} \otimes \mathbf{I} + (|1\rangle\langle 1|)_{2q} \otimes \mathbf{X}] \right) \mathbf{M}(\mathcal{S}_{(q+1)}^{-1}) \\
&= \mathbf{M}(\mathcal{S}_{(q+1)}) \left(\sum_{k=0}^{2^q-1} |k\rangle\langle k| \otimes \bigotimes_{j=0}^{q-1} \mathbf{X}^{b(k)_j} \otimes |0\rangle\langle 0|_{2q} \otimes \mathbf{I} \right) \mathcal{S}_{(q)} \\
&\quad + \mathcal{S}_{(q)}^{-1} \left(\sum_{k=0}^{2^q-1} |k\rangle\langle k| \otimes \bigotimes_{j=0}^{q-1} \mathbf{X}^{b(k)_j} \otimes |1\rangle\langle 1|_{2q} \otimes \mathbf{X} \right) \mathbf{M}(\mathcal{S}_{(q+!)}^{-1}) \\
&= \sum_{k=0}^{2^q-1} |k\rangle\langle k| \otimes |0\rangle\langle 0|_{2q} \otimes \bigotimes_{j=0}^{q-1} \mathbf{X}^{b(k)_j} \otimes \mathbf{I} + \sum_{k=0}^{2^q-1} |k\rangle\langle k| \otimes (|1\rangle\langle 1|)_{2q} \otimes \bigotimes_{j=0}^{q-1} \mathbf{X}^{b(k)_j} \otimes \mathbf{X} \\
&= \sum_{k=0}^{2^q-1} |2k\rangle\langle 2k| \otimes \bigotimes_{j=0}^{q} \mathbf{X}^{b(2k)_j} + \sum_{k=0}^{2^q-1} |2k+1\rangle\langle 2k+1| \bigotimes_{j=0}^{q} \mathbf{X}^{b(2k+1)_j} \\
&= \sum_{k'=0}^{2^{q+1}-1} |k'\rangle\langle k'| \otimes \bigotimes_{j=0}^{q} \mathbf{X}^{b(k')_j} \\
&= \sum_{k'=0}^{2^{q+1}-1} |k'\rangle\langle k'| \otimes \mathbf{M}(\mathcal{U}_{k'})
\end{aligned}$$

where we used the fact that for any set of M single qubit gates $\mathcal{U}_0, \ldots, \mathcal{U}_{M-1}$ we have: (See Figure 5 for a visualization)

$$\mathcal{S}_\sigma \cdot \left(\bigotimes_{i=0}^{M-1} \mathcal{U}_i \right) \cdot \mathcal{S}_{\sigma^{-1}} = \bigotimes_{i=0}^{M-1} \mathcal{U}_{\sigma(i)}$$

which in terms of matrices associated with the circuit translates to

$$\mathbf{M}(\mathcal{S}_\sigma) \cdot \left(\bigotimes_{i=0}^{M-1} \mathbf{M}(\mathcal{U}_i) \right) \cdot \mathbf{M}(\mathcal{S}_{\sigma^{-1}}) = \bigotimes_{i=0}^{M-1} \mathbf{M}(\mathcal{U}_{\sigma(i)})$$

□

Fact 22. *For each* $\mathbf{w} \in \Sigma^q$,

$$N\alpha_{\mathbf{A}}(\mathbf{w}) = \mathbf{vec}\left(\sigma_{\mathbf{w}}^T\right)^T \mathbf{vec}\left(\mathbf{A}\right)$$

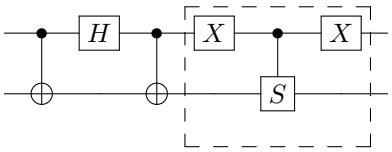


Figure 6: Circuit of $\mathcal{U}^{(1)}$. The dashed box highlights the diagonal circuit $\mathcal{D}$.

Proof. We have,

$$\begin{aligned} N\alpha_{\mathbf{A}}(\mathbf{w}) &= \mathbf{Tr}\left(\sigma_{\mathbf{w}}\mathbf{A}\right) \\ &= \mathbf{cvec}\left(\sigma_{\mathbf{w}}^{\mathrm{T}}\right)^{*}\mathbf{cvec}\left(\mathbf{A}^{\mathrm{T}}\right) \\ &= \mathbf{vec}\left(\sigma_{\mathbf{w}}\right)^{*}\mathbf{vec}\left(\mathbf{A}\right) \\ &= \overline{\mathbf{vec}\left(\sigma_{\mathbf{w}}\right)^{\mathrm{T}}}\mathbf{vec}\left(\mathbf{A}\right) \\ &= \mathbf{vec}\left(\overline{\sigma_{\mathbf{w}}}\right)^{\mathrm{T}}\mathbf{vec}\left(\mathbf{A}\right) \\ &= \mathbf{vec}\left(\sigma_{\mathbf{w}}^{\mathrm{T}}\right)^{\mathrm{T}}\mathbf{vec}\left(\mathbf{A}\right) \end{aligned}$$

where the last line equality follows from the fact that for an Hermitian matrix $\mathbf{X}$ we have $\overline{\mathbf{X}} = \mathbf{X}^{\mathrm{T}}$. □

Lemma 23. *Consider the circuit,*

$$\mathcal{U}^{(q)} = \mathcal{D}_{(q)} \cdot \mathcal{P}_{(q)} \cdot \mathcal{U}_{\mathrm{comb}}^{(q)} \cdot \left(\mathcal{H}^{\otimes q} \otimes \mathcal{I}_q\right) \cdot \mathcal{U}_{\mathrm{comb}}^{(q)}$$

where $\mathcal{D}_{(q)} = \bigotimes_{i=0}^{q}\mathcal{D}$ *with* $\mathbf{M}(\mathcal{D}) = \mathbf{diag}\left(1,1,i,1\right)$*,* $\mathcal{P}_{(q)}$ *is a circuit that interlaces qubits of two size* q *registers, that is if the original qubits are labeled* $(0,1,\ldots,q-1,\ q,q+1,\ldots,2q-1)$ *then after* $\mathcal{P}_{(q)}$ *the qubits are ordered* $(0,q,1,q+1,\ldots,q-1,2q-1)$*, and* $\mathcal{H}^{\otimes q}$ *denotes the* q*-qubit circuit that has an Hadamard gate on each qubit in parallel. See Figure 7 for a visualization of the circuit. Then the rows of the matrix* $\mathbf{M}\left(\mathcal{U}^{(q)}\right)$ *are exactly the vectorizations of the* q-wise Pauli matrices $\sigma_{\mathbf{w}_0},\ldots,\sigma_{\mathbf{w}_{4^q-1}}$*, in lexicographic order:*

$$\mathbf{M}\left(\mathcal{U}^{(q)}\right) = \frac{1}{\sqrt{2^q}}\begin{bmatrix} \mathbf{vec}\left(\sigma_{\mathbf{w}_0}^{T}\right)^{T} \\ \mathbf{vec}\left(\sigma_{\mathbf{w}_1}^{T}\right)^{T} \\ \vdots \\ \mathbf{vec}\left(\sigma_{\mathbf{w}_{4^q-1}}^{T}\right)^{T} \end{bmatrix}.$$

Proof. We prove by induction on q. For $q = 1$ one checks directly that (see Figure 6):

$$\begin{aligned}
\mathcal{U}^{(1)} &= \frac{1}{\sqrt{2}}\mathcal{D}\begin{bmatrix} \mathbf{I}_2 & \\ & \mathbf{X} \end{bmatrix}\begin{bmatrix} \mathbf{I}_2 & \mathbf{I}_2 \\ \mathbf{I}_2 & -\mathbf{I}_2 \end{bmatrix}\begin{bmatrix} \mathbf{I}_2 & \\ & \mathbf{X} \end{bmatrix} \\
&= \frac{1}{\sqrt{2}}\mathbf{diag}\,(1,1,\mathrm{i},1)\begin{bmatrix} \mathbf{I} & \mathbf{X} \\ \mathbf{X} & -\mathbf{I} \end{bmatrix} \\
&= \frac{1}{\sqrt{2}}\mathbf{diag}\,(1,1,\mathrm{i},1)\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & -1 \end{bmatrix} \\
&= \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & \mathrm{i} & -\mathrm{i} & 0 \\ 1 & 0 & 0 & -1 \end{bmatrix} \\
&= \frac{1}{\sqrt{2}}\begin{bmatrix} \mathbf{vec}\,(\mathbf{I}_2)^{\mathrm{T}} \\ \mathbf{vec}\,(\mathbf{X})^{\mathrm{T}} \\ \mathbf{vec}\left(\mathbf{Y}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\,(\mathbf{Z})^{\mathrm{T}} \end{bmatrix}
\end{aligned}$$

which agrees with the four one-qubit Paulis $(\sigma_{\mathbf{w}_0}, \ldots, \sigma_{\mathbf{w}_3}) = (I, X, Y, Z)$ in lexicographic order.

Assume the statement holds for q, i.e.,

$$\mathbf{M}\big(\mathcal{U}^{(q)}\big) = \frac{1}{\sqrt{2^q}}\begin{bmatrix} \mathbf{vec}\left(\sigma_{w_0}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \vdots \\ \mathbf{vec}\left(\sigma_{w_{4^q-1}}^{\mathrm{T}}\right)^{\mathrm{T}} \end{bmatrix}.$$

We first make the following observation. Let $\mathcal{V}^{(q)} \coloneqq \mathcal{U}^{(q)}_{\mathrm{comb}} \cdot \big(\mathcal{H}^{\otimes q} \otimes \mathcal{I}_q\big) \cdot \mathcal{U}^{(q)}_{\mathrm{comb}}$, so that by definition $\mathcal{U}^{(q)} = \mathcal{D}_{(q)} \cdot \mathcal{P}_{(q)} \cdot \mathcal{V}^{(q)}$. We have the following recursive relation:

$$\begin{aligned}
\mathbf{M}(\mathcal{V}^{(1)}) &= \mathbf{M}(\mathcal{U}^{(1)}_{\mathrm{comb}} \cdot \big(\mathcal{H} \otimes \mathcal{I}_1\big) \cdot \mathcal{U}^{(1)}_{\mathrm{comb}}) \\
&= |0\rangle\langle 0| \otimes \mathbf{I} + |0\rangle\langle 1| \otimes \mathbf{X} + |1\rangle\langle 0| \otimes \mathbf{X} - |1\rangle\langle 1| \otimes \mathbf{I}
\end{aligned}$$

and

$$\mathcal{V}^{(q+1)} = \mathcal{S}_{(q+1)} \cdot \Big(\mathcal{V}^{(q)} \otimes \mathcal{V}^{(1)}\Big) \cdot \mathcal{S}^{-1}_{(q+1)}$$

Indeed, we have

$$\begin{aligned}
\mathcal{V}^{(q+1)} &= \mathcal{U}^{(q+1)}_{\mathrm{comb}} \cdot \big(\mathcal{H}^{\otimes q+1} \otimes \mathcal{I}_{q+1}\big) \cdot \mathcal{U}^{(q+1)}_{\mathrm{comb}} \\
&= \Big[\mathcal{S}_{(q+1)} \cdot (\mathcal{U}^{(q)}_{\mathrm{comb}} \otimes \mathrm{CNOT}_{2q,2q+1}) \cdot \mathcal{S}^{-1}_{(q+1)}\Big] \cdot \Big[\mathcal{S}_{(q+1)} \cdot (\mathcal{H}^{\otimes q} \otimes \mathcal{I}_q \otimes \mathcal{H} \otimes \mathcal{I}_1) \cdot \mathcal{S}^{-1}_{(q+1)}\Big] \cdot \Big[\mathcal{S}_{(q+1)} \cdot (\mathcal{U}^{(q)}_{\mathrm{comb}} \otimes \mathrm{CNOT}_{2q,2q+1}) \cdot \mathcal{S}^{-1}_{(q+1)}\Big] \\
&= \mathcal{S}_{(q+1)} \cdot (\mathcal{U}^{(q)}_{\mathrm{comb}} \otimes \mathrm{CNOT}_{2q,2q+1}) \cdot (\mathcal{H}^{\otimes q} \otimes \mathcal{I}_q \otimes \mathcal{H} \otimes \mathcal{I}_1) \cdot (\mathcal{U}^{(q)}_{\mathrm{comb}} \otimes \mathrm{CNOT}_{2q,2q+1}) \cdot \mathcal{S}^{-1}_{(q+1)} \\
&= \mathcal{S}_{(q+1)} \cdot \Big((\mathcal{U}^{(q)}_{\mathrm{comb}} \cdot (\mathcal{H}^{\otimes q} \otimes \mathcal{I}_q) \cdot \mathcal{U}^{(q)}_{\mathrm{comb}}) \otimes (\mathrm{CNOT}_{2q,2q+1} \cdot (\mathcal{H} \otimes \mathcal{I}_1) \cdot \mathrm{CNOT}_{2q,2q+1})\Big) \cdot \mathcal{S}^{-1}_{(q+1)} \\
&= \mathcal{S}_{(q+1)} \cdot \Big(\mathcal{V}^{(q)} \otimes \mathcal{V}^{(1)}\Big) \cdot \mathcal{S}^{-1}_{(q+1)}
\end{aligned}$$

where we used the Proposition 21 and $\mathcal{H}^{\otimes q+1} \otimes \mathcal{I}_{q+1} = \mathcal{S}_{(q+1)} \cdot ((\mathcal{H}^{\otimes q} \otimes \mathcal{I}_q) \otimes (\mathcal{H} \otimes \mathcal{I}_1)) \cdot \mathcal{S}^{-1}_{(q+1)}$.

The next observation is that:

$$\mathcal{P}_{(q+1)} = (\mathcal{P}_{(q)} \otimes \mathcal{I}_1) \cdot \mathcal{S}^{-1}_{(q+1)}$$

where $\mathcal{S}_{(q+1)}$ is defined in the previous proposition. Proof of this observation is deferred to the end, in order not to disturb the flow of the proof. The last equation implies that,

$$\begin{aligned}
\mathcal{U}^{(q+1)} &= \mathcal{D}_{(q+1)} \cdot \mathcal{P}_{(q+1)} \cdot \mathcal{V}^{(q+1)} \\
&= \mathcal{D}_{(q+1)} \cdot (\mathcal{P}_{(q)} \otimes \mathcal{I}_1) \cdot \mathcal{S}_{(q+1)}^{-1} \cdot \mathcal{S}_{(q+1)} \cdot \left(\mathcal{V}^{(q)} \otimes V^{(1)}\right) \cdot \mathcal{S}_{(q+1)}^{-1} \\
&= \mathcal{D}_{(q+1)} \cdot (\mathcal{P}_{(q)} \mathcal{V}^{(q)} \otimes \mathcal{I}_1 \mathcal{V}^{(1)}) \cdot \mathcal{S}_{(q+1)}^{-1} \\
&= (D_{(q)} \otimes \mathcal{D}) \cdot \left(\mathcal{P}_{(q)} \mathcal{V}^{(q)} \otimes \mathcal{V}^{(1)}\right) \cdot \mathcal{S}_{(q+1)}^{-1} \\
&= (\mathcal{U}^{(q)} \otimes \mathcal{U}^{(1)}) \cdot \mathcal{S}_{(q+1)}^{-1}
\end{aligned}$$

The following is a known identity: for every $\mathbf{A} \in \mathbb{C}^{N \times M}$ and $\mathbf{B} \in^{L \times K}$ we have

$$\mathbf{cvec}\left(\mathbf{A} \otimes \mathbf{B}\right) = \left(\mathbf{I}_M \otimes \mathbf{P}_{K,N} \otimes \mathbf{I}_L\right) \left[\mathbf{cvec}\left(\mathbf{A}\right) \otimes \mathbf{cvec}\left(\mathbf{B}\right)\right]$$

where $\mathbf{cvec}\left(\cdot\right)$ denotes column-major vectorization of a matrix, and $\mathbf{P}_{N,M}$ is the Kronecker permutation matrix.

$$\mathbf{P}_{N,M} = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \mathbf{E}_{ij}^{(N \times M)} \otimes \mathbf{E}_{ji}^{(M \times N)}$$

(see [7, Fact 7.4.29.(xii)]). Translating to row-major vectorization, we have

$$\mathbf{vec}\left(\mathbf{A}^{\mathrm{T}} \otimes \mathbf{B}^{\mathrm{T}}\right) = \left(\mathbf{I}_M \otimes \mathbf{P}_{K,N} \otimes \mathbf{I}_L\right) \left[\mathbf{vec}\left(\mathbf{A}^{\mathrm{T}}\right) \otimes \mathbf{vec}\left(\mathbf{B}^{\mathrm{T}}\right)\right]$$

In our case, we have for any 2-by-2 Pauli matrix σ_i and for any q-wise Pauli matrix $\sigma_{\mathbf{w}} \in \mathbb{C}^{N \times N}$ (note that in the following, we are using transpose and not conjugate transpose):

$$\mathbf{vec}\left(\sigma_{\mathbf{w}}^{\mathrm{T}} \otimes \sigma_i^{\mathrm{T}}\right) = \left(\mathbf{I}_N \otimes \mathbf{P}_{2,N} \otimes \mathbf{I}_2\right) \left[\mathbf{vec}\left(\sigma_{\mathbf{w}}^{\mathrm{T}}\right) \otimes \mathbf{vec}\left(\sigma_i^{\mathrm{T}}\right)\right]$$

So, we have

$$
\begin{aligned}
\mathbf{M}(\mathcal{U}^{(q+1)}) &= \left(\mathbf{M}(\mathcal{U}^{(q)}) \otimes \mathbf{M}(\mathcal{U}^{(1)})\right) \mathbf{M}(\mathcal{S}^{-1}_{(q+1)}) \\
&= \left(\frac{1}{\sqrt{2^q}} \begin{bmatrix} \mathbf{vec}\left(\sigma^{\mathrm{T}}_{w_0}\right)^{\mathrm{T}} \\ \mathbf{vec}\left(\sigma^{\mathrm{T}}_{w_1}\right)^{\mathrm{T}} \\ \vdots \\ \mathbf{vec}\left(\sigma^{\mathrm{T}}_{w_{4^q-1}}\right)^{\mathrm{T}} \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{vec}\left(\mathbf{I}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left(\mathbf{X}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left(\mathbf{Y}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left(\mathbf{Z}^{\mathrm{T}}\right)^{\mathrm{T}} \end{bmatrix} \right) \mathbf{M}(\mathcal{S}^{-1}_{(q+1)}) \\
&= \frac{1}{\sqrt{2^{q+1}}} \begin{bmatrix} \mathbf{vec}\left(\sigma^{\mathrm{T}}_{\mathbf{w}_0}\right)^{\mathrm{T}} \otimes \mathbf{vec}\left(\mathbf{I}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left(\sigma^{\mathrm{T}}_{\mathbf{w}_0}\right)^{\mathrm{T}} \otimes \mathbf{vec}\left(\mathbf{X}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left(\sigma^{\mathrm{T}}_{\mathbf{w}_0}\right)^{\mathrm{T}} \otimes \mathbf{vec}\left(\mathbf{Y}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left(\sigma^{\mathrm{T}}_{\mathbf{w}_0}\right)^{\mathrm{T}} \otimes \mathbf{vec}\left(\mathbf{Z}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left(\sigma^{\mathrm{T}}_{\mathbf{w}_1}\right)^{\mathrm{T}} \otimes \mathbf{vec}\left(\mathbf{I}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \vdots \end{bmatrix} \mathbf{M}(\mathcal{S}^{-1}_{(q+1)}) \\
&= \frac{1}{\sqrt{2^{q+1}}} \left(\mathbf{M}(\mathcal{S}_{(q+1)}) \begin{bmatrix} \mathbf{vec}\left(\sigma^{\mathrm{T}}_{\mathbf{w}_0}\right)^{\mathrm{T}} \otimes \mathbf{vec}\left(\mathbf{I}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left(\sigma^{\mathrm{T}}_{\mathbf{w}_0}\right)^{\mathrm{T}} \otimes \mathbf{vec}\left(\mathbf{X}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left(\sigma^{\mathrm{T}}_{\mathbf{w}_0}\right)^{\mathrm{T}} \otimes \mathbf{vec}\left(\mathbf{Y}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left(\sigma^{\mathrm{T}}_{\mathbf{w}_0}\right)^{\mathrm{T}} \otimes \mathbf{vec}\left(\mathbf{Z}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left(\sigma^{\mathrm{T}}_{\mathbf{w}_1}\right)^{\mathrm{T}} \otimes \mathbf{vec}\left(\mathbf{I}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \vdots \end{bmatrix}^{\mathrm{T}} \right)^{\mathrm{T}} \\
&= \frac{1}{\sqrt{2^{q+1}}} \left(\mathbf{I}_N \otimes \mathbf{P}_{2,N} \otimes \mathbf{I}_2 \begin{bmatrix} \mathbf{vec}\left(\sigma^{\mathrm{T}}_{\mathbf{w}_0}\right)^{\mathrm{T}} \otimes \mathbf{vec}\left(\mathbf{I}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left(\sigma^{\mathrm{T}}_{\mathbf{w}_0}\right)^{\mathrm{T}} \otimes \mathbf{vec}\left(\mathbf{X}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left(\sigma^{\mathrm{T}}_{\mathbf{w}_0}\right)^{\mathrm{T}} \otimes \mathbf{vec}\left(\mathbf{Y}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left(\sigma^{\mathrm{T}}_{\mathbf{w}_0}\right)^{\mathrm{T}} \otimes \mathbf{vec}\left(\mathbf{Z}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left(\sigma^{\mathrm{T}}_{\mathbf{w}_1}\right)^{\mathrm{T}} \otimes \mathbf{vec}\left(\mathbf{I}^{\mathrm{T}}\right)^{\mathrm{T}} \\ \vdots \end{bmatrix}^{\mathrm{T}} \right)^{\mathrm{T}} \\
&= \frac{1}{\sqrt{2^{q+1}}} \begin{bmatrix} \mathbf{vec}\left((\sigma_{\mathbf{w}_0} \otimes \mathbf{I})^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left((\sigma_{\mathbf{w}_0} \otimes \mathbf{X})^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left((\sigma_{\mathbf{w}_0} \otimes \mathbf{Y})^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left((\sigma_{\mathbf{w}_0} \otimes \mathbf{Z})^{\mathrm{T}}\right)^{\mathrm{T}} \\ \mathbf{vec}\left((\sigma_{\mathbf{w}_1} \otimes \mathbf{I})^{\mathrm{T}}\right)^{\mathrm{T}} \\ \vdots \\ \mathbf{vec}\left((\sigma^{\mathrm{T}}_{\mathbf{w}_{4^q-1}} \otimes \mathbf{Z})^{\mathrm{T}}\right)^{\mathrm{T}} \end{bmatrix}
\end{aligned}
$$

where the fifth equality follows from Proposition 19.

Finally we need to complete the argument $\mathcal{P}_{(q+1)} = (\mathcal{P}_{(q)} \otimes \mathcal{I}_1) \cdot \mathcal{S}^{-1}_{(q+1)}$. It is easy to see that we have $\mathcal{P}_{(q)} = \mathcal{S}_{\tau(q)}$ where $\tau_{(q)} \in S_{2q}$ is the following permutation:

$$
\tau_{(q)}(i) = \begin{cases} 2i & \text{if } 0 \le i < q \\ 2(i-q)+1 & \text{if } q \le i < 2q \end{cases}
$$

The permutation $\tau_{(q+1)}$ can be decomposed as $\tau_{(q+1)} = (\tau_{(q)} \oplus \mathrm{id}_1) \circ (\sigma^{(q+1)})^{-1}$, which implies that

$$
\mathcal{P}_{(q+1)} = (\mathcal{P}_{(q)} \otimes \mathcal{I}_1) \cdot \mathcal{S}^{-1}_{(q+1)}
$$

as required. To see that $\tau_{(q+1)} = (\tau_{(q)} \oplus \mathrm{id}_1) \circ (\sigma^{(q+1)})^{-1}$ we show that both permutations produce identical results when applied to any index $i \in \{0, 1, \ldots, 2q+1\}$.

- For $i \in \{0, 1, \ldots, q-1\}$: since $i < q$, we have $(\sigma^{(q+1)})^{-1}(i) = i$. Thus:

$$\begin{aligned}(\tau_{(q)} \oplus \mathrm{id}_1)(\sigma^{(q+1)})^{-1})(i) &= (\tau_{(q)} \oplus \mathrm{id}_1)(i) \\ &= \tau_{(q)}(i) \quad (\text{since } i < 2q) \\ &= 2i\end{aligned}$$

  On the other hand, since $i < q < q+1$ we have, by definition:

$$\tau_{(q+1)}(i) = 2i$$

- For $i = q$: we have $(\sigma^{(q+1)})^{-1}(q) = 2q$, so since $2q \geq 2q$ we have

$$\begin{aligned}(\tau_{(q)} \oplus \mathrm{id}_1)(\sigma^{(q+1)})^{-1})(q) &= (\tau_{(q)} \oplus \mathrm{id}_1)(2q) \\ &= 2q \quad (\text{since } \tau_{(q)} \oplus \mathrm{id}_1 \text{ fixes } 2q)\end{aligned}$$

  On the other hand, since $q < q+1$:

$$\tau_{(q+1)}(q) = 2q$$

- For $i \in \{q+1, q+2, \ldots, 2q\}$: we have $(\sigma^{(q+1)})^{-1}(i) = i-1$, so since $q \leq i-1 < 2q$:

$$\begin{aligned}(\tau_{(q)} \oplus \mathrm{id}_1)(\sigma^{(q+1)})^{-1})(i) &= (\tau_{(q)} \oplus \mathrm{id}_1)(i-1) \\ &= \tau_{(q)}(i-1) \quad (\text{since } i-1 < 2q) \\ &= 2((i-1)-q)+1 \quad (\text{since } i-1 \geq q) \\ &= 2i-2q-1\end{aligned}$$

  On the other hand, since $i \geq q+1$:

$$\begin{aligned}\tau_{(q+1)}(i) &= 2(i-(q+1))+1 \\ &= 2i-2q-1\end{aligned}$$

- For $i = 2q+1$: since $i = 2q+1 > 2q$ we have $(\sigma^{(q+1)})^{-1}(2q+1) = 2q+1$, and so

$$\begin{aligned}((\tau_{(q)} \oplus \mathrm{id}_1)(\sigma^{(q+1)})^{-1})(2q+1) &= (\tau_{(q)} \oplus \mathrm{id}_1)(2q+1) \\ &= 2q+1 \quad (\text{since } (\tau_{(q)} \oplus \mathrm{id}_1) \text{ fixes } 2q+1)\end{aligned}$$

  On the other hand, since $2q+1 \geq q+1$:

$$\begin{aligned}\tau_{(q+1)}(2q+1) &= 2((2q+1)-(q+1))+1 \\ &= 2q+1\end{aligned}$$

This covers all cases. Since the two permutations are equal for all indices, the two permutations are equal. □

Remark 24. In order to create $\mathcal{D}$ such that $\mathbf{M}(\mathcal{D}) = \mathbf{diag}(1, 1, \mathrm{i}, 1)$ we note that

$$\mathbf{M}(\mathcal{D}) = \begin{bmatrix} \mathbf{I} & \\ & \mathbf{XSX} \end{bmatrix} = \begin{bmatrix} \mathbf{X} & \\ & \mathbf{X} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \\ & \mathbf{S} \end{bmatrix} \begin{bmatrix} \mathbf{X} & \\ & \mathbf{X} \end{bmatrix}$$

See Figure 6 for a circuit diagram of $\mathcal{D}$.

We are now ready to state the main theorem of this section.

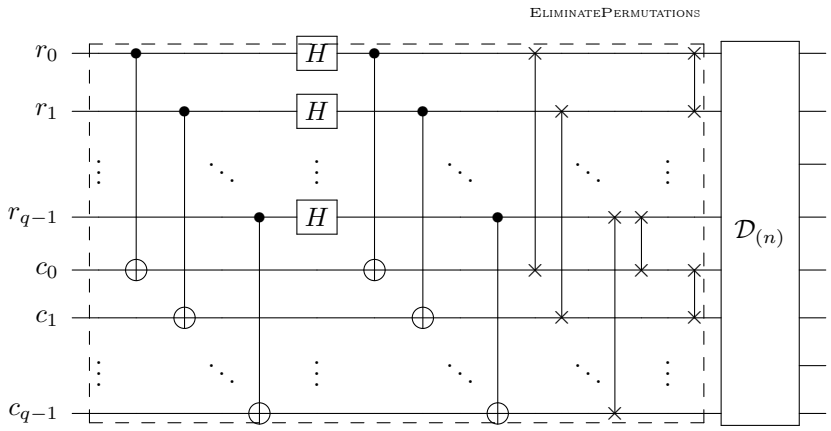


Figure 7: Visualization of the $\mathcal{U}^{(q)} = \mathcal{D}_{(q)} \cdot \mathcal{P}_{(q)} \cdot \mathcal{U}^{(q)}_{\text{comb}} \cdot \left(\mathcal{H}^{\otimes q} \otimes \mathcal{I}_q\right) \cdot \mathcal{U}^{(q)}_{\text{comb}}$. In order to use the ELIMINATE-PERMUTATIONS optimization, the final circuit must be a state preparation circuit, meaning it must act on the ground state.

Theorem 25. Let $\mathbf{A} \in \mathbb{C}^{N \times N}$ where $N = 2^n$. Given a classical description of a matrix state preparation circuit $\mathcal{U}^{\text{SP}}_{\mathbf{A}} \in \mathbf{MS}_\alpha(\mathbf{A})$, let $\mathcal{U}^{\text{SP}}_{\mathcal{A}_P} \coloneqq (\mathcal{I}_{q(\mathcal{U}_{\mathbf{A}})-2n} \otimes \mathcal{U}^{(n)}) \cdot \mathcal{U}^{\text{SP}}_{\mathbf{A}}$ (see Lemma 23 for $\mathcal{U}^{(n)}$'s definition; if $q(\mathcal{U}^{\text{SP}}_{\mathbf{A}}) = 2n$ then then identity part is omitted); see Figure 8 for a circuit diagram. The circuit $\mathcal{U}^{\text{SP}}_{\mathcal{A}_P}$ is a hypermatrix state preparation for the Pauli coefficient hypermatrix $\mathcal{A}_P$. In particular, $\mathcal{U}^{\text{SP}}_{\mathcal{A}_P} \in \mathbf{HS}_{\alpha/\sqrt{N}}(\mathcal{A}_P)$. The circuit $\mathcal{U}^{\text{SP}}_{\mathcal{A}_P}$ can be constructed with classical cost $O(g(\mathcal{U}^{\text{SP}}_{\mathbf{A}}) + n)$, and with $g(\mathcal{U}^{\text{SP}}_{\mathbf{A}}) + 6n$ gate complexity. This construction yields a depth of $d(\mathcal{U}^{\text{SP}}_{\mathbf{A}}) + 5$, and with surplus T-gate and T-depth (compared to the T-costs of $\mathcal{U}^{\text{SP}}_{\mathbf{A}}$) is n and 1 (respectively).

Proof. From Lemma 23 the matrix $\mathbf{M}(\mathcal{U}^{(n)})$ consists of rows that are the vectorized (and transposed) Pauli operators:

$$\mathbf{M}(\mathcal{U}^{(n)}) = \frac{1}{\sqrt{N}} \begin{bmatrix} \mathbf{vec}\left(\sigma^{\text{T}}_{\mathbf{w}_0}\right)^{\text{T}} \\ \mathbf{vec}\left(\sigma^{\text{T}}_{\mathbf{w}_1}\right)^{\text{T}} \\ \vdots \\ \mathbf{vec}\left(\sigma^{\text{T}}_{\mathbf{w}_{4^n-1}}\right)^{\text{T}} \end{bmatrix}$$

Since $\mathcal{U}^{\text{SP}}_{\mathbf{A}} \in \mathbf{MS}_\alpha(\mathbf{A})$, there exists a θ such that

$$\alpha e^{\text{i}\theta} \mathbf{M}(\mathcal{U}^{\text{SP}}_{\mathbf{A}}) = \begin{bmatrix} \vdots & * & \cdots & * \\ \mathbf{vec}(\mathbf{A}) & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \boldsymbol{\psi} & \vdots & & \vdots \\ \vdots & * & \cdots & * \end{bmatrix}$$

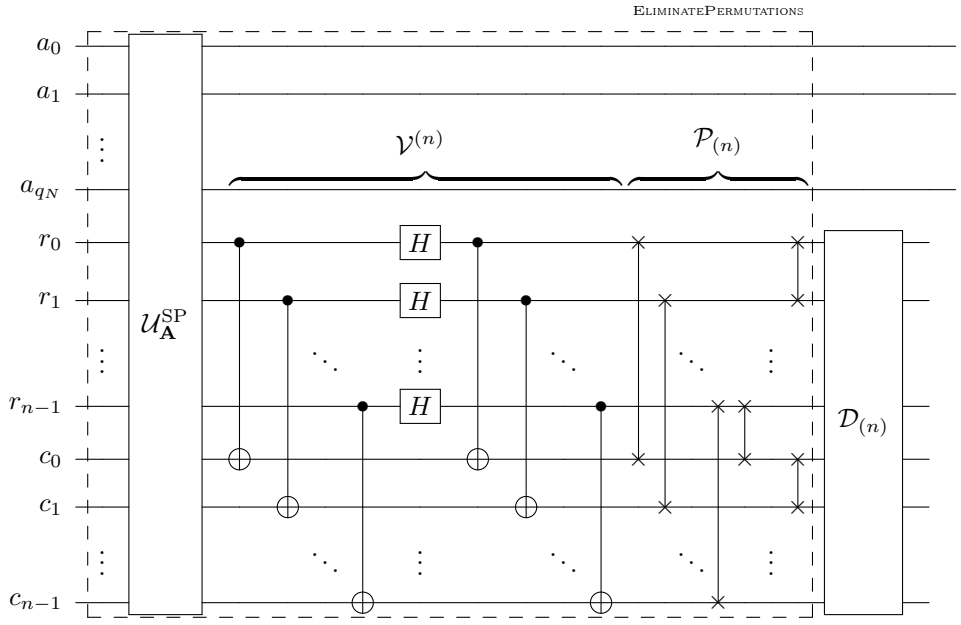


Figure 8: Circuit of $\mathcal{U}_{\mathcal{A}_P}^{\mathrm{SP}}$. The top $q_N \coloneqq q(\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}}) - 2n$ wires correspond to ancillary qubits on which only $\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}}$ acts. The bottom $2n$ wires are the data qubits, where after applying $\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}}$ the sequence of circuits $\mathcal{V}^{(n)}$, $\mathcal{P}_{(n)}$, and $\mathcal{D}_{(n)}$ are applied.

where $\boldsymbol{\psi} \in \mathbb{C}^{2^{q(\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}})-2n}}$. So,

$$\alpha e^{\mathrm{i}\theta}\mathbf{M}(\mathcal{I}_{q(\mathcal{U}_{\mathbf{A}})-2n} \otimes \mathcal{U}^{(n)})\mathbf{M}(\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}}) = \begin{bmatrix} \mathbf{M}(\mathcal{U}^{(n)}) & 0 & \cdots \\ 0 & \ddots & 0 \\ \vdots & 0 & \mathbf{M}(\mathcal{U}^{(n)}) \end{bmatrix} \begin{bmatrix} \vdots & * & \cdots & * \\ \mathbf{vec}\,(\mathbf{A}) & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \boldsymbol{\psi} & \vdots & & \vdots \\ \vdots & * & \cdots & * \end{bmatrix}$$

$$= \sqrt{N} \begin{bmatrix} \alpha_{\mathbf{A}}(\mathbf{w}_0) & * & \cdots & * \\ \vdots & \vdots & & \vdots \\ \alpha_{\mathbf{A}}(\mathbf{w}_{4^{q-1}}) & \vdots & & \vdots \\ * & \vdots & & \vdots \\ \vdots & & & \\ * & * & \cdots & * \end{bmatrix}$$

where in the last equality we used the identity $N\alpha_{\mathbf{A}}(\mathbf{w}) = \mathbf{vec}\left(\sigma_{\mathbf{w}}^{\mathrm{T}}\right)^{\mathrm{T}} \mathbf{vec}\,(\mathbf{A})$. Dividing by $\sqrt{N}$ on both sides, we see that $\mathcal{U}_{\mathcal{A}_P}^{\mathrm{SP}} \in \mathbf{HS}_{\alpha/\sqrt{N}}(\mathcal{A}_P)$ by definition of $\mathbf{HS}_{\alpha/\sqrt{N}}(\mathcal{A}_P)$.

Using EliminatePermutations procedure for $\mathcal{P}_{(n)} \cdot \mathcal{U}_{\text{comb}}^{(n)} \cdot \left(\mathcal{H}^{\otimes n} \otimes \mathcal{I}_n\right) \cdot \mathcal{U}_{\text{comb}}^{(n)} \cdot \mathcal{U}_{\mathbf{A}}^{\text{SP}}$ to eliminate $\mathcal{P}_{(n)}$, we have that the depth of $\mathcal{U}_{\mathcal{A}_p}^{\text{SP}}$ is $d(\mathcal{U}_{\mathbf{A}}^{\text{SP}}) + 6$ the gate complexity and the classical cost are $g(\mathcal{U}_{\mathbf{A}}^{\text{SP}}) + 6n$ (see Figure 8). □

By reversing the process, we can convert from a state preparation of $\mathcal{A}_P$ to a state preparation of $\mathbf{A}$. To see this, note that if

$$\mathcal{U}_{\mathcal{A}_P}^{\text{SP}} = \mathcal{U}^{(n)} \cdot \mathcal{U}_{\mathbf{A}}^{\text{SP}}$$

then

$$\mathcal{U}_{\mathbf{A}}^{\text{SP}} = (\mathcal{U}^{(n)})^* \cdot \mathcal{U}_{\mathcal{A}_P}^{\text{SP}}$$

where $(\mathcal{U}^{(n)})^*$ denotes the inverse of the circuit. The scaling factor is adjusted by $\sqrt{N}$.

Remark 26. Seemingly moving from a state preparation of $\mathbf{A}$ to a state preperation of $\mathcal{A}_P$ improves the scale (makes it smaller) by a factor of $1/\sqrt{N}$. However, the Frobenius norm of $\|\mathcal{A}_P\|_F$ is also smaller by a factor of $1/\sqrt{N}$. Since only the ratio between the scale and the Frobenius norm of the prepared hypermatrix matters, there is no gain in terms of the scale for moving from state preparation of $\mathcal{A}_P$ to state preparation of $\mathbf{A}$. Similarly, when moving from $\mathcal{A}_P$ to $\mathbf{A}$ the scale "worsens" by a factor of $\sqrt{N}$, however, there is no actual loss here as well since the Frobenius norm increases by the same factor. Moving one direction and then back around keeps the scale the same, and only marginally increases the depth. So the two modes of state preparation are effectively equivalent.

5 Multiplexing higher-order Pauli matrices

The algorithms we propose for constructing block encodings employ multiplexers of higher-order Pauli matrices. This section describes how to construct such multiplexers efficiently.

5.1 Constructing a multiplexer for arbitrary set of higher-order Pauli matrices

In this subsection, our goal is that given a length K sequence $\mathbf{w}_0, \ldots, \mathbf{w}_{K-1}$ of length q Pauli words (i.e., $\mathbf{w}_j \in \Sigma^q$), construct an efficient circuit $\mathcal{PMX}(\mathbf{w}_0, \ldots, \mathbf{w}_{K-1}) \in \mathcal{MX}(\mathcal{V}_{\mathbf{w}_0}, \ldots, \mathcal{V}_{\mathbf{w}_{K-1}})$, where $\mathcal{V}_{\mathbf{w}}$ implements the higher order Pauli matrix corresponding to the word $\mathbf{w}$. That is, the circuit $\mathcal{PMX}(\mathbf{w}_0, \ldots, \mathbf{w}_{K-1})$ is constructed such that

$$\mathbf{M}(\mathcal{PMX}(\mathbf{w}_0, \ldots, \mathbf{w}_{K-1})) = \bigoplus_{j=0}^{K-1} \sigma_{\mathbf{w}_j}. \tag{6}$$

Define the rotation matrices:

$$\mathbf{R}_y(\theta) \coloneqq \begin{bmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix}, \ \mathbf{R}_z(\theta) \coloneqq \begin{bmatrix} e^{-\mathrm{i}\frac{\theta}{2}} & 0 \\ 0 & e^{\mathrm{i}\frac{\theta}{2}} \end{bmatrix}.$$

These are the operators that correspond to the $\mathcal{R}_y$ and $\mathcal{R}_z$ rotation gates. The key to our approach is the well known fact that we can write every Pauli matrix as the product of specific combination of $\mathbf{R}_y$ and $\mathbf{R}_z$ rotation matrices:

$$\begin{aligned} \mathbf{I} &= \mathbf{R}_y(0)\mathbf{R}_z(0) \\ -\mathrm{i}\mathbf{X} &= \mathbf{R}_y(\pi)\mathbf{R}_z(\pi) \\ -\mathrm{i}\mathbf{Y} &= \mathbf{R}_y(\pi)\mathbf{R}_z(0) \\ -\mathrm{i}\mathbf{Z} &= \mathbf{R}_y(0)\mathbf{R}_z(\pi) \end{aligned} \tag{7}$$

Going from Pauli matrices to higher-order Pauli matrices, given a Pauli string $\mathbf{w}$ we can find a set of angles $\{\theta_i^{(y)}\}_{i=0}^{q-1}$ and $\{\theta_i^{(z)}\}_{i=0}^{q-1}$, all of them 0 or π, and a phase φ such that

$$e^{-\mathrm{i}\varphi}\sigma_{\mathbf{w}} = \bigotimes_{i=0}^{q-1} \mathbf{R}_y(\theta_i^{(y)})\mathbf{R}_z(\theta_i^{(z)})$$

The phase φ corresponds to $\pi/2$ times the number number of X, Y, Z in $\mathbf{w}$. Thus, given words $\mathbf{w}_0, \ldots, \mathbf{w}_{K-1}$, there exists angles $\{\theta_{ij}^{(y)}\}$ and $\{\theta_{ij}^{(z)}\}$ (where i runs from 0 to $q-1$, and j from 0 to $K-1$), and phases $\varphi_0, \ldots, \varphi_{K-1}$, such that

$$\bigoplus_{j=0}^{K-1} e^{-\mathrm{i}\varphi_j}\sigma_{\mathbf{w}_j} = \bigoplus_{j=0}^{K-1}\bigotimes_{i=0}^{q-1} \mathbf{R}_y(\theta_{ij}^{(y)})\mathbf{R}_z(\theta_{ij}^{(z)}) \tag{8}$$

Both $\mathbf{R}_y$ and $\mathbf{R}_z$ rotation matrices can be implemented using the corresponding rotation gates, $\mathcal{R}_y$ for $\mathbf{R}_y$ and $\mathcal{R}_z$ for $\mathbf{R}_z$. This allows us to implement a multiplexer of higher-order Pauli matrices, up to per element scaling, using a sequence of two 2D multiplexers, first a $\mathcal{R}_z$ multiplexer, and then a $\mathcal{R}_y$ multiplexers. The key is the following identity: given matrices $\mathbf{A}_{ij}, \mathbf{B}_{ij}$ for $i = 0, \ldots L-1, j = 0, \ldots, M-1$ of compatible sizes, we have

$$\begin{aligned}
\bigoplus_{i=0}^{L-1}\bigotimes_{j=0}^{M-1}(\mathbf{A}_{ij}\mathbf{B}_{ij}) &= \bigoplus_{i=0}^{L-1}\left[\left(\bigotimes_{j=0}^{M-1}\mathbf{A}_{ij}\right)\left(\bigotimes_{j=0}^{M-1}\mathbf{B}_{ij}\right)\right] \\
&= \left(\bigoplus_{i=0}^{L-1}\bigotimes_{j=0}^{M-1}\mathbf{A}_{ij}\right)\left(\bigoplus_{i=0}^{L-1}\bigotimes_{j=0}^{M-1}\mathbf{B}_{ij}\right)
\end{aligned}$$

where the first equality follows mixed-product property of Kronecker products, and the second one is product of block diagonal matrices. Applying this identity to Eq. (8), we have

$$\bigoplus_{j=0}^{K-1} e^{-\mathrm{i}\varphi_j}\sigma_{\mathbf{w}_j} = \left(\bigoplus_{j=0}^{K-1}\bigotimes_{i=0}^{q-1}\mathbf{R}_y(\theta_{ij}^{(y)})\right)\left(\bigoplus_{j=0}^{K-1}\bigotimes_{i=0}^{q-1}\mathbf{R}_z(\theta_{ij}^{(z)})\right) \tag{9}$$

All angles in the two multiplexers are either 0 or π. Finally, the phases can be corrected by a diagonal gate with diagonal values

$$\underbrace{\mathbf{e}^{\mathrm{i}\varphi_0}, \ldots, \mathbf{e}^{\mathrm{i}\varphi_0}}_{q \text{ times}}, \underbrace{\mathbf{e}^{\mathrm{i}\varphi_1}, \ldots, \mathbf{e}^{\mathrm{i}\varphi_1}}_{q \text{ times}}, \ldots, \underbrace{\mathbf{e}^{\mathrm{i}\varphi_{K-1}}, \ldots, \mathbf{e}^{\mathrm{i}\varphi_{K-1}}}_{q \text{ times}}$$

Such a diagonal matrix is the Kronecker product of the identity matrix with the diagonal matrix with distinct phases, so we need only to implement a diagonal gate for the phases themselves. Thus, the diagonal gate for correcting the phases, using a generic implementation, has depth complexity of $O(K/k)$ and T-gate and T-depth of $O(Kk)$ and $O(K)$ respectively.

Using the generic rotation multiplexer construction described in Section 3 we can construct a multiplexer for any set of Pauli words of the same length. Since for K Pauli words of length q we need two 2D multiplexers of size q-by-K, the number of single qubit rotations in the multiplexers is $O(qK)$, number of fanout-CNOTs is $O(K)$, and depth is $O(K)$. Combined with the costs associated with the diagonal gate, we have total depth of $O(Kq)$. However, in subsequent sections we make use of multiplexers for higher order Pauli matrices corresponding to words of a given length. As we will see in the next subsection, such multiplexers can be implemented using highly efficient circuits.

5.2 Multiplexing all higher-order Pauli matrices

In this section we consider the construction of a multiplexer for all Pauli words in $\mathbb{P}_N$. The construction in the previous section can be used to that end, i.e., all we need is to impose some order on the Pauli words, $\mathbf{w}_0, \ldots, \mathbf{w}_{N^2-1}$, and form $\mathcal{PMX}(\mathbf{w}_0, \ldots, \mathbf{w}_{N^2-1})$. However, we shall show that if we choose a specific order, we can construct a highly efficient circuit, $\mathcal{PMX}_n$, which implements a multiplexer of all Pauli words.

Fix n, and let $N = 2^n$. The ordering on $\mathbb{P}_N$ that we use to implement the multiplexer is the lexicographical ordering on the Pauli words (most important index is the first, i.e., on the left). This corresponds to viewing the Pauli words as integers written in the base-4, with the mapping $0 \longleftrightarrow I, 1 \longleftrightarrow X, 2 \longleftrightarrow Y, 3 \longleftrightarrow Z$.

For presentation purposes, it is useful to define an auxiliary matrix $\mathbf{P}_n \in \Sigma^{n \times N^2}$ as follows. Each column in $\mathbf{P}_n$ corresponds to a Pauli word, according to the lexicographical order. For a column corresponding to word $\mathbf{w}$, we map the individual letters to rows, with left being on the top. For example,

$$\mathbf{P}_1 = \begin{bmatrix} I & X & Y & Z \end{bmatrix}$$

$$\mathbf{P}_2 = \begin{bmatrix} I & I & I & I & X & X & X & X & Y & Y & Y & Y & Z & Z & Z & Z \\ I & X & Y & Z & I & X & Y & Z & I & X & Y & Z & I & X & Y & Z \end{bmatrix}$$

$$\mathbf{P}_3 = \begin{bmatrix} I & I & I & I & I & I & I & I & I & I & I & I & I & I & I & I & \cdots \\ I & I & I & I & X & X & X & X & Y & Y & Y & Y & Z & Z & Z & Z & \cdots \\ I & X & Y & Z & I & X & Y & Z & I & X & Y & Z & I & X & Y & Z & \cdots \end{bmatrix}$$

A recursive formula for $\mathbf{P}_n$ is as follows:

$$\mathbf{P}_n = \begin{bmatrix} \underbrace{I\,I\,\cdots\,I}_{4^{n-1}\text{ times}} & \underbrace{X\,X\,\cdots\,X}_{4^{n-1}\text{ times}} & \underbrace{Y\,Y\,\cdots\,Y}_{4^{n-1}\text{ times}} & \underbrace{Z\,Z\,\cdots\,Z}_{4^{n-1}\text{ times}} \\ \mathbf{P}_{n-1} & \mathbf{P}_{n-1} & \mathbf{P}_{n-1} & \mathbf{P}_{n-1} \end{bmatrix} \tag{10}$$

Assume that $\mathbf{w}_0, \ldots, \mathbf{w}_{N^2-1}$ are all the Pauli words, ordered according to the lexicographical order. We already seen in the previous section that there exists matrices $\mathbf{\Theta}_n^{(P,y)} = \left[\theta_{ij}^{(P,y)}\right]_{n \times N^2}$ and $\mathbf{\Theta}_n^{(P,z)} = \left[\theta_{ij}^{(P,z)}\right]_{n \times N^2}$ and angles $\{\varphi_j^{(P)}\}$ (the superscript P denotes the use for multiplexing all Pauli matrices) such that

$$\bigoplus_{j=0}^{4^n-1} e^{\mathrm{i}\varphi_j^{(P)}} \sigma_{\mathbf{w}_j} = \left(\bigoplus_{j=0}^{4^n-1} \bigotimes_{i=0}^{n-1} \mathbf{R}_y(\theta_{ij}^{(P,y)}) \right) \left(\bigoplus_{j=0}^{4^n-1} \bigotimes_{i=0}^{n-1} \mathbf{R}_z(\theta_{ij}^{(P,z)}) \right)$$

The entries matrix $\mathbf{\Theta}^{(P,y)}$ and $\mathbf{\Theta}^{(P,z)}$ are easily defined by the corresponding entries of $\mathbf{P}_n$:

$$\theta_{ij}^{(P,z)} = \begin{cases} \pi & (\mathbf{P}_n)_{ij} = X \text{ or } Z \\ 0 & (\mathbf{P}_n)_{ij} = I \text{ or } Y \end{cases} \qquad \theta_{ij}^{(P,y)} = \begin{cases} \pi & (\mathbf{P}_n)_{ij} = X \text{ or } Y \\ 0 & (\mathbf{P}_n)_{ij} = I \text{ or } Z \end{cases}$$

From the recursive formula for $\mathbf{P}_n$ (Eq. (10)), we have the following recursive formula for $\mathbf{\Theta}^{(P,y)}$ and $\mathbf{\Theta}^{(P,z)}$:

$$\mathbf{\Theta}_1^{(P,z)} = \begin{bmatrix} 0 & \pi & 0 & \pi \end{bmatrix} \quad \text{and} \quad \Theta_1^{(P,y)} = \begin{bmatrix} 0 & \pi & \pi & 0 \end{bmatrix}$$

$$\mathbf{\Theta}_n^{(P,z)} = \begin{bmatrix} \underbrace{0\cdots0}_{4^{n-1}\text{ times}} & \underbrace{\pi\cdots\pi}_{4^{n-1}\text{ times}} & \underbrace{0\cdots0}_{4^{n-1}\text{ times}} & \underbrace{\pi\cdots\pi}_{4^{n-1}\text{ times}} \\ \mathbf{\Theta}_{n-1}^{(P,z)} & \mathbf{\Theta}_{n-1}^{(P,z)} & \mathbf{\Theta}_{n-1}^{(P,z)} & \mathbf{\Theta}_{n-1}^{(P,z)} \end{bmatrix}$$

$$\mathbf{\Theta}_n^{(P,y)} = \begin{bmatrix} \underbrace{0\cdots0}_{4^{n-1}\text{ times}} & \underbrace{\pi\cdots\pi}_{4^{n-1}\text{ times}} & \underbrace{\pi\cdots\pi}_{4^{n-1}\text{ times}} & \underbrace{0\cdots0}_{4^{n-1}\text{ times}} \\ \mathbf{\Theta}_{n-1}^{(P,y)} & \mathbf{\Theta}_{n-1}^{(P,y)} & \mathbf{\Theta}_{n-1}^{(P,y)} & \mathbf{\Theta}_{n-1}^{(P,y)} \end{bmatrix}$$

For example,

$$\mathbf{\Theta}_2^{(P,z)} = \begin{bmatrix} 0 & 0 & 0 & 0 & \pi & \pi & \pi & \pi & 0 & 0 & 0 & 0 & \pi & \pi & \pi & \pi \\ 0 & \pi & 0 & \pi & 0 & \pi & 0 & \pi & 0 & \pi & 0 & \pi & 0 & \pi & 0 & \pi \end{bmatrix}$$

$$\mathbf{\Theta}_2^{(P,y)} = \begin{bmatrix} 0 & 0 & 0 & 0 & \pi & \pi & \pi & \pi & \pi & \pi & \pi & \pi & 0 & 0 & 0 & 0 \\ 0 & \pi & \pi & 0 & 0 & \pi & \pi & 0 & 0 & \pi & \pi & 0 & 0 & \pi & \pi & 0 \end{bmatrix}$$

5.2.1 Ultra-sparsity of $\hat{\mathbf{\Theta}}_n^{(P,y)}$ and $\hat{\mathbf{\Theta}}_n^{(P,z)}$

Key to an efficient multiplexer for all Pauli matrices is the ultra sparsity of $\hat{\mathbf{\Theta}}_n^{(P,y)}$ and $\hat{\mathbf{\Theta}}_n^{(P,z)}$. Recall that when using matrix $\mathbf{\Theta}$ with M columns in a rotation multiplexer, we set up an array of rotation gates with CNOTs between them, where the angles are given by $\hat{\mathbf{\Theta}} = \frac{1}{\sqrt{M}}\mathbf{\Theta}\mathbf{H}_M\mathbf{G}_M$ with $\mathbf{H}_M$ being the $M \times M$ Walsh-Hadamard matrix, and $\mathbf{G}_M$ is the permutation matrix that transforms binary ordering to Gray code ordering (see Fact 9). For multiplexing all Pauli words, for the two multiplexers we have $\hat{\mathbf{\Theta}}_n^{(P,y)} = \frac{1}{N}\mathbf{\Theta}_n^{(P,y)}\mathbf{H}_{N^2}\mathbf{G}_{N^2}$ and $\hat{\mathbf{\Theta}}_n^{(P,z)} = \frac{1}{N}\mathbf{\Theta}_n^{(P,z)}\mathbf{H}_{N^2}\mathbf{G}_{N^2}$. The following proposition shows that these two matrices are ultra sparse and contain only $\pm\pi/2$:

Proposition 27. *The following holds: (recall that we use* 0 *based indexing)*

1. All entries in the first column of $\hat{\mathbf{\Theta}}_n^{(P,y)}$ and $\hat{\mathbf{\Theta}}_n^{(P,z)}$ have value $\pi/2$.
2. For $\hat{\mathbf{\Theta}}_n^{(P,z)}$, each row contains one other non-zero. At row i it is in column $2 \cdot 4^{n-i-1} - 1$ with value $-\pi/2$.
3. For $\hat{\mathbf{\Theta}}_n^{(P,y)}$, each row contains one other non-zero. At row i it is in column $2 \cdot 4^{n-i-1}$ with value $-\pi/2$.

Proof. For the first column, note that the each row in $\mathbf{\Theta}_n^{(P,z)}$ and $\mathbf{\Theta}_n^{(P.y)}$ consists of exactly $N^2/2$ entries non-zero entries, each of them equal to π. The first column of $\mathbf{H}_{N^2/2}\mathbf{G}_{N^2}$ is the all ones vector divided by N, since this the value of the first column in the Walsh-Hadamard matrix and BRGC does not change the location of the first index. So, the first column in $\hat{\mathbf{\Theta}}_n^{(P,z)}$ and $\hat{\mathbf{\Theta}}_n^{(P,y)}$ is equal to the sum of entries in the rows of $\mathbf{\Theta}_n^{(P,z)}$ and $\mathbf{\Theta}_n^{(P,y)}$ divided by N^2. Together, we see that the value is indeed equal to $\pi/2$.

For other non-zero entries, we prove using induction. The base case follows from simple calculations:

$$\begin{aligned}
\hat{\mathbf{\Theta}}_1^{(P,z)} &= \frac{1}{\sqrt{4}}\begin{bmatrix} 0 & \pi & 0 & \pi \end{bmatrix} \underbrace{\left(\frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \right)}_{\mathbf{H}_4} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{G}_4} \\
&= \frac{1}{4}\begin{bmatrix} 0 & \pi & 0 & \pi \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \\
&= \begin{bmatrix} \pi/2 & -\pi/2 & 0 & 0 \end{bmatrix}
\end{aligned}$$

Likewise,

$$\hat{\mathbf{\Theta}}_1^{(P,y)} = \frac{1}{4}\begin{bmatrix} 0 & \pi & \pi & 0 \end{bmatrix}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$
$$= \begin{bmatrix} \pi/2 & 0 & -\pi/2 & 0 \end{bmatrix}$$

For the inductive step, we use the following well known formula for $\mathbf{G}_M$ when M is a power of 2. Let $\mathbf{J}_M$ denote the M-by-M index reversal matrix, i.e., the M-by-M matrix with 1 on the anti-diagonal. Then, (see [45, Section 5])

$$\mathbf{G}_M = \mathbf{G}_{M/2} \oplus \mathbf{G}_{M/2}\mathbf{J}_{M/2}$$

Thus, we have

$$\mathbf{G}_{4^n} = \begin{bmatrix} \mathbf{G}_{4^n/2} & \mathbf{0} \\ \mathbf{0} & \mathbf{G}_{4^n/2}\mathbf{J}_{4^n/2} \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{G}_{4^{n-1}} & & & \\ & \mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} & & \\ & & \mathbf{G}_{4^{n-1}} & \\ & & & \mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} \end{bmatrix}\begin{bmatrix} \mathbf{I}_{4^{n-1}} & & & \\ & \mathbf{I}_{4^{n-1}} & & \\ & & & \mathbf{J}_{4^{n-1}} \\ & & \mathbf{J}_{4^{n-1}} & \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{G}_{4^{n-1}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} \\ \mathbf{0} & \mathbf{0} & \mathbf{G}_{4^{n-1}} & \mathbf{0} \end{bmatrix}$$

using the fact that $\mathbf{J}_M^2 = \mathbf{I}_M$. The recursive formula for for $\mathbf{H}_{4^n}$ implies that:

$$\mathbf{H}_{4^n} = \frac{1}{\sqrt{2}}\begin{bmatrix} \mathbf{H}_{4^n/2} & \mathbf{H}_{4^n/2} \\ \mathbf{H}_{4^n/2} & -\mathbf{H}_{4^n/2} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} \mathbf{H}_{4^{n-1}} & \mathbf{H}_{4^{n-1}} & \mathbf{H}_{4^{n-1}} & \mathbf{H}_{4^{n-1}} \\ \mathbf{H}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}} & \mathbf{H}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}} \\ \mathbf{H}_{4^{n-1}} & \mathbf{H}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}} \\ \mathbf{H}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}} & \mathbf{H}_{4^{n-1}} \end{bmatrix}$$

So,

$$\hat{\mathbf{\Theta}}_n^{(P,z)} = \frac{1}{N}\Theta_n^{(z)}\mathbf{H}_{4^n}\mathbf{G}_{4^n} = \frac{1}{2^{n+1}}\begin{bmatrix} \mathbf{0}_{1\times 4^{n-1}} & \pi\mathbf{e}_{1\times 4^{n-1}} & \mathbf{0}_{1\times 4^{n-1}} & \pi\mathbf{e}_{1\times 4^{n-1}} \\ \mathbf{\Theta}_{n-1}^{(P,z)} & \mathbf{\Theta}_{n-1}^{(P,z)} & \mathbf{\Theta}_{n-1}^{(P,z)} & \mathbf{\Theta}_{n-1}^{(P,z)} \end{bmatrix}$$
$$\cdot \begin{bmatrix} \mathbf{H}_{4^{n-1}} & \mathbf{H}_{4^{n-1}} & \mathbf{H}_{4^{n-1}} & \mathbf{H}_{4^{n-1}} \\ \mathbf{H}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}} & \mathbf{H}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}} \\ \mathbf{H}_{4^{n-1}} & \mathbf{H}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}} \\ \mathbf{H}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}} & \mathbf{H}_{4^{n-1}} \end{bmatrix}\begin{bmatrix} \mathbf{G}_{4^{n-1}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} \\ \mathbf{0} & \mathbf{0} & \mathbf{G}_{4^{n-1}} & \mathbf{0} \end{bmatrix}$$
$$= \frac{1}{2^{n+1}}\begin{bmatrix} \mathbf{0}_{1\times 4^{n-1}} & \pi\mathbf{e}_{1\times 4^{n-1}} & \mathbf{0}_{1\times 4^{n-1}} & \pi\mathbf{e}_{1\times 4^{n-1}} \\ \mathbf{\Theta}_{n-1}^{(P,z)} & \mathbf{\Theta}_{n-1}^{(P,z)} & \mathbf{\Theta}_{n-1}^{(P,z)} & \mathbf{\Theta}_{n-1}^{(P,z)} \end{bmatrix}$$
$$\cdot \begin{bmatrix} \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} & \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} \\ \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} \\ \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} \\ \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} & \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} \end{bmatrix}$$
$$= \frac{1}{2^{n+1}}\begin{bmatrix} 2\pi\mathbf{e}_{1\times 4^{n-1}}\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & -2\pi\mathbf{e}_{1\times 4^{n-1}}\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} & \mathbf{0}_{1\times 4^{n-1}} & \mathbf{0}_{1\times 4^{n-1}} \\ 4\Theta_{n-1}^{(P,z)}\mathbf{H}_{4^{n-1}} & \mathbf{0}_{1\times 4^{n-1}} & \mathbf{0}_{1\times 4^{n-1}} & \mathbf{0}_{1\times 4^{n-1}} \end{bmatrix}$$
$$= \frac{1}{2^{n+1}}\begin{bmatrix} \underbrace{2\pi\sqrt{4^{n-1}} \quad 0 \quad \cdots \quad 0}_{4^{n-1}\text{ length}} & \underbrace{0 \quad \cdots \quad 0 \quad -2\pi\sqrt{4^{n-1}}}_{4^{n-1}\text{ length}} & \underbrace{0 \quad \cdots \quad 0}_{4^{n-1}\text{ length}} & \underbrace{0 \quad \cdots \quad 0}_{4^{n-1}\text{ length}} \\ 4\mathbf{\Theta}_{n-1}^{(P,z)}\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$
$$= \begin{bmatrix} \underbrace{\pi/2 \quad 0 \quad \cdots \quad 0}_{4^{n-1}\text{ length}} & \underbrace{0 \quad \cdots \quad 0 \quad -\pi/2}_{4^{n-1}\text{ length}} & \underbrace{0 \quad \cdots \quad 0}_{4^{n-1}\text{ length}} & \underbrace{0 \quad \cdots \quad 0}_{4^{n-1}\text{ length}} \\ \frac{1}{2^{n-1}}\mathbf{\Theta}_{q-1}^{(P,z)}\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

In the above, we used the fact that for any M which is a power of 2, we have $\mathbf{e}_{1\times M}\mathbf{H}_M = \begin{bmatrix} \sqrt{M} & 0 & \cdots & 0 \end{bmatrix}$ (this follows from the fact that the first column of $\sqrt{M}\mathbf{H}_M$ is the all ones vector, while other columns are balanced in the number of $+1$ and -1). This implies that $\mathbf{e}_{1\times M}\mathbf{H}_M\mathbf{G}_{4^{n-1}} = \begin{bmatrix} \sqrt{M} & 0 & \cdots & 0 \end{bmatrix}$ and $\mathbf{e}_{1\times M}\mathbf{H}_M\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} = \begin{bmatrix} 0 & \cdots & 0 & \sqrt{M} \end{bmatrix}$. We see that the claim holds for the first row, i.e., index $i = 0$. The claim holds for the rest of the rows by induction, that gives the structure of $\frac{1}{2^{n-1}}\mathbf{\Theta}^{(z)}_{n-1}\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}$.

The proof for $\hat{\mathbf{\Theta}}^{(P,y)}_n$ follows the same logic, as follows:

$$
\begin{aligned}
\hat{\mathbf{\Theta}}^{(P,y)}_n = \frac{1}{N}\Theta^{(y)}_n\mathbf{H}_{4^n}\mathbf{G}_{4^n} &= \frac{1}{2^{n+1}}\begin{bmatrix} \mathbf{0}_{1\times 4^{n-1}} & \pi\mathbf{e}_{1\times 4^{n-1}} & \pi\mathbf{e}_{1\times 4^{n-1}} & \mathbf{0}_{1\times 4^{n-1}} \\ \mathbf{\Theta}^{(P,y)}_{n-1} & \mathbf{\Theta}^{(P,y)}_{n-1} & \mathbf{\Theta}^{(P,y)}_{n-1} & \mathbf{\Theta}^{(P,y)}_{n-1} \end{bmatrix} \\
&\quad \cdot \begin{bmatrix} \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} & \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} \\ \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} \\ \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} \\ \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} & \mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & -\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}}\mathbf{J}_{4^{n-1}} \end{bmatrix} \\
&= \frac{1}{2^{n+1}}\begin{bmatrix} 2\pi\mathbf{e}_{1\times n}\mathbf{H}_{4^{n-1}}\mathbf{G}_{4q-1} & \mathbf{0}_{1\times 4^{n-1}} & -2\pi\mathbf{e}_{1\times 4^{n-1}}\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & \mathbf{0}_{1\times 4^{n-1}} \\ 4\Theta^{(P.y)}_{n-1}\mathbf{H}_{4^{n-1}} & \mathbf{0}_{1\times 4^{n-1}} & \mathbf{0}_{1\times 4^{n-1}} & \mathbf{0}_{1\times 4^{n-1}} \end{bmatrix} \\
&= \frac{1}{2^{n+1}}\begin{bmatrix} \underbrace{2\pi\sqrt{4^{n-1}} \quad 0 \quad \cdots \quad 0}_{4^{n-1}\text{ length}} & \underbrace{0 \quad \cdots \quad 0}_{4^{n-1}\text{ length}} & \underbrace{-2\pi\sqrt{4^{n-1}} \quad 0 \quad \cdots \quad 0}_{} & \underbrace{0 \quad \cdots \quad 0}_{4^{n-1}\text{ length}} \\ 4\mathbf{\Theta}^{(P.y)}_{n-1}\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \\
&= \begin{bmatrix} \underbrace{\pi/2 \quad 0 \quad \cdots \quad 0}_{4^{n-1}\text{ length}} & \underbrace{0 \quad \cdots \quad 0}_{4^{n-1}\text{ length}} & \underbrace{-\pi/2 \quad 0 \quad \cdots \quad 0}_{4^{n-1}\text{ length}} & \underbrace{0 \quad \cdots \quad 0}_{4^{n-1}\text{ length}} \\ \frac{1}{2^{n-1}}\mathbf{\Theta}^{(P.y)}_{n-1}\mathbf{H}_{4^{n-1}}\mathbf{G}_{4^{n-1}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}
\end{aligned}
$$

□

5.2.2 Ultra low depth rotation multiplexers for $\mathbf{\Theta}^{(P,y)}_n$ and $\mathbf{\Theta}^{(P,z)}_n$

By Proposition 27, the angles matrices $\hat{\mathbf{\Theta}}^{(P,z)}_n$ and $\hat{\mathbf{\Theta}}^{(P,y)}_n$ are ultra sparse (each row has only two nonzero entries $\pm\pi/2$). In this section, we leverage this to construct two families of efficient circuits, $2d\mathcal{PMUX}_{y,n}$ and $2d\mathcal{PMUX}_{z,n}$ (where $n = 1, 2, \ldots$ is a parameter), such that

$$2d\mathcal{PMUX}_{y,n} \in \mathcal{MX}_y(\mathbf{\Theta}^{(P,y)}_n), \quad 2d\mathcal{PMUX}_{z,n} \in \mathcal{MX}_z(\mathbf{\Theta}^{(P,z)}_n) \tag{11}$$

Derivation of $2d\mathcal{PMUX}_{y,n}$ and $2d\mathcal{PMUX}_{z,n}$ is based on two well known circuit rewriting rules:

Fact 28. *Consider a circuit $\mathcal{U}$ which contains either a $\mathcal{R}_y$ or $\mathcal{R}_z$ gate with angle 0. Then the circuit obtained by erasing that gate from $\mathcal{U}$ is equivalent to $\mathcal{U}$.*

Fact 29. *Consider a circuit $\mathcal{U}$. Suppose that in $\mathcal{U}$ there are two CNOTs with the same target and control, and in which between them there are only CNOTs with the same target and different control, or gates that operate on neither the target or control. Then the circuit $\mathcal{U}'$ obtained by removing the two aforementioned CNOTs from $\mathcal{U}$ is equivalent to $\mathcal{U}$.*

We now show that repeated application of these two facts starting with $2d\mathcal{MR}_y(\mathbf{\Theta}^{(P,y)}_n) \in \mathcal{MX}_y(\mathbf{\Theta}^{(P,y)}_n)$ and $2d\mathcal{MR}_z(\mathbf{\Theta}^{(P,z)}_n) \in \mathcal{MX}_z(\mathbf{\Theta}^{(P,z)}_z)$ results in the following circuits, which are our proposed $2d\mathcal{PMUX}_{y,n}$ and $2d\mathcal{PMUX}_{z,n}$. The circuits $2d\mathcal{PMUX}_{y,n}$ and $2d\mathcal{PMUX}_{z,n}$ have $3n$ qubits. Of these, n are target qubits, and are labeled $t_0, \ldots, t_{n-1}$ (each corresponding to a row in $\mathbf{\Theta}^{(P,\cdot)}_n$), and the rest are control qubits labeled $c_0, \ldots, c_{2n-1}$. On each target we operate with rotation gates, or with CNOTs with one or two specific control qubits. Each control qubit is connected to exactly one target qubit, and some control qubits are empty. Thus, we can describe the circuits by just detailing the gates on each target qubit. These description is as follows:

- **Target qubit t_i in $2d\mathcal{PMAX}_{z,n}$, $i = 0, 1, \ldots, n-1$:**

$$2d\mathcal{PMAX}_{z,n}\Big|_{t_i} = \mathcal{R}_{z,t_i}(\pi/2) \cdot \mathrm{CX}(c_{2i+1}, t_i) \cdot \mathcal{R}_{z,t_i}(-\pi/2) \cdot \mathrm{CX}(c_{2i+1}, t_i). \tag{12}$$

- **Target qubit t_i in $2d\mathcal{PMAX}_{y,n}$, $i = 0, 1, \ldots, n-1$:**

$$2d\mathcal{PMAX}_{y,n}\Big|_{t_i} = \mathcal{R}_{y,t_i}(\pi/2) \cdot \mathrm{CX}(c_{2i+1}, t_i) \cdot \mathrm{CX}(c_{2i}, t) \cdot \mathcal{R}_{y,t_i}(-\pi/2) \cdot \mathrm{CX}(c_{2i+1}, t_i) \cdot \mathrm{CX}(c_{2i}, t_i). \tag{13}$$

See Figure 9 for a graphical illustration. Note that each data qubit t_j is controlled by one or both of $\{c_{2j}, c_{2j+1}\}$, and that the total number of CNOTs is $\#\mathrm{CNOT}(2d\mathcal{PMAX}_{z,n}) = 2n$ and $\#\mathrm{CNOT}(2d\mathcal{PMAX}_{z,n}) = 4n$. Clearly, the gate count is $O(n)$ (we have $2n$ rotation angles and $2n$ or $4n$ CNOT gates for z or y, respectively). The circuit depth is $O(1)$ (each qubit has 2 rotations and at most 2 or 4 CNOT gates, so the critical path is at most 4 or 6). The T-count and T-depth are 0 (rotations with angles $\pm\pi/2$ are Clifford gates). The classical cost construction is $O(n)$ (no angle computations are needed since all rotations are $\pm\pi/2$).

We now prove that these constructed circuits indeed implement the desired multiplexers, i.e., Eqs. (11) hold. We start with two auxiliary lemmas. (See Section 3.1 for the definition of $\mathcal{F}_m^{(d)}$.)

Lemma 30. *Fix m. For any depth $d \in \{0, \ldots, m-1\}$,*

$$\mathcal{F}_m^{(d)}(\underbrace{0, \ldots, 0}_{2^{m-d} \text{ angles}}) \cong CX(c_d, t).$$

Proof. By induction on $m - d$. Base case is $m - d = 1$, or $d = m - 1$, and we have (Eq. (3))

$$\mathcal{F}_m^{(m-1)}(0, 0) = \mathcal{R}_y(0) \cdot \mathrm{CX}(c_{m-1}, t) \cdot \mathcal{R}_y(0) = \mathrm{CX}(c_{m-1}, t)$$

Next, we consider the inductive case, i.e., $m - d$ is increased by 1 (so d is decreased by 1, since we consider m fixed). The inductive assumption translates to

$$\mathcal{F}_m^{(d+1)}(\underbrace{0, \ldots, 0}_{2^{m-(d+1)} \text{ angles}}) \cong \mathrm{CX}(c_{d+1}, t)$$

We have

$$\begin{aligned}\mathcal{F}_m^{(d)}(\underbrace{0, \ldots, 0}_{2^{m-d} \text{ angles}}) &= \mathcal{F}_m^{(d+1)}(0, \ldots, 0) \cdot \mathrm{CX}(c_d, t) \cdot \mathcal{F}_m^{(d+1)}(0, \ldots, 0) \\ &\cong \mathrm{CX}(c_{d+1}, t) \cdot \mathrm{CX}(c_d, t) \cdot \mathrm{CX}(c_{d+1}, t) \\ &\cong \mathrm{CX}(c_d, t)\end{aligned}$$

where we used Eq. (2). □

Lemma 31. *Fix m. For any depth $d \in \{0, \ldots, m-1\}$,*

1. $\mathcal{F}_m^{(d)}\big(\phi_0, \underbrace{0, \ldots, 0}_{2^{m-d}-1}\big) \cong \mathcal{R}(\phi_0) \cdot CX(c_d, t)$.
2. $\mathcal{F}_m^{(d)}\big(\underbrace{0, \ldots, 0}_{2^{m-d}-1}, \phi_{2^{m-d}-1}\big) \cong CX(c_d, t) \cdot \mathcal{R}(\phi_{2^{m-d}-1})$.

Proof. By induction on $m-d$. Base case is $m-d=1$, or $d=m-1$, and we have (Eq. (3))

$$\mathcal{F}_m^{(m-1)}(\phi_0,\phi_1)=\mathcal{R}_y(\phi_0)\cdot \mathrm{CX}(c_{m-1},t)\cdot\mathcal{R}_y(\phi_1)=\begin{cases}\mathcal{R}_y(\phi_0)\cdot \mathrm{CX}(c_{m-1},t) & \text{if } \phi_1=0\\ \mathrm{CX}(c_{m-1},t)\cdot\mathcal{R}_y(\phi_1) & \text{if } \phi_0=0\end{cases}$$

Now we consider the inductive case, i.e., $m-d$ is increased by 1 (so d is decreased by 1, since we consider m fixed). The inductive assumption translates to

$$\mathcal{F}_m^{(d+1)}(\phi_0,0,\ldots,0)\cong\mathcal{R}(\phi_0)\cdot\mathrm{CX}(c_{d+1},t)$$
$$\mathcal{F}_m^{(d+1)}(0,\ldots,0,\phi_{2^{m-(d+1)}-1})\cong\mathrm{CX}(c_{d+1},t)\cdot\mathcal{R}(\phi_{2^{m-(d+1)}-1})$$

We show for the prefix case (first identity). The proof for suffix case (second identity) follows the same logic. We have,

$$\begin{aligned}\mathcal{F}_m^{(d)}(\phi_0,0,\ldots,0)&=\underbrace{\mathcal{F}_m^{(d+1)}(\phi_0,0,\ldots,0)}_{\mathcal{R}(\phi_0)\cdot\mathrm{CX}(c_{d+1},t)}\cdot\mathrm{CX}(c_d,t)\cdot\underbrace{\mathcal{F}_m^{(d+1)}(0,\ldots,0)}_{\mathrm{CNOT}(c_{d+1},t)}\\&\cong\mathcal{R}(\phi_0)\cdot\mathrm{CX}(c_{d+1},t)\cdot\mathrm{CX}(c_d,t)\cdot\mathrm{CX}(c_{d+1},t)\\&\cong\mathcal{R}(\phi_0)\cdot\mathrm{CX}(c_d,t)\end{aligned}$$

where we used Eq. (2) and Lemma 30. □

Proposition 32. *We have,*

$$2d\mathcal{PMA}_{y,n}\in\mathcal{MA}_y(\boldsymbol{\Theta}_n^{(P,y)}),\quad 2d\mathcal{PMA}_{z,n}\in\mathcal{MA}_z(\boldsymbol{\Theta}_n^{(P,z)})$$

Equivalently:

$$\mathbf{M}(2d\mathcal{PMA}_{y,n})=\bigoplus_{j=0}^{N^2-1}\bigotimes_{i=0}^{n-1}\mathbf{R}_y\big(\theta_{ij}^{(P,y)}\big),\qquad \mathbf{M}(2d\mathcal{PMA}_{z,n})=\bigoplus_{j=0}^{N^2-1}\bigotimes_{i=0}^{n-1}\mathbf{R}_z\big(\theta_{ij}^{(P,z)}\big).$$

Proof. The idea is to start with $2d\mathcal{MAR}_y(\boldsymbol{\Theta}_n^{(P,y)})$ and $2d\mathcal{MAR}_z(\boldsymbol{\Theta}_n^{(P,z)})$ and via repeated applications of Lemmas 30 and 31 obtain smaller and smaller equivalent circuits, until reaching $2d\mathcal{PMA}_{y,n}$ and $2d\mathcal{PMA}_{z,n}$. Put differently, we show that $2d\mathcal{PMA}_{y,n}\Big|_{t_i}\cong 2d\mathcal{MAR}_y(\boldsymbol{\Theta}_n^{(P,y)})\Big|_{t_i}$ and $2d\mathcal{PMA}_{z,n}\Big|_{t_i}\cong 2d\mathcal{MAR}_z(\boldsymbol{\Theta}_n^{(P,z)})\Big|_{t_i}$ for $i=0,\ldots,n-1$, and this implies the result.

Consider the base case of $n=1$. We have $\hat{\boldsymbol{\Theta}}_1^{(P,z)}=[+\pi/2,-\pi/2,0,0]$, and $\hat{\boldsymbol{\Theta}}_1^{(P,y)}=[+\pi/2,0,-\pi/2,0]$. The fact that the claim holds is straightforward (see Figure 2).

Assume that the claim holds for n, i.e., $2d\mathcal{MAR}_\xi(\boldsymbol{\Theta}_n^{(P,\xi)})\Big|_{t_i}\cong 2d\mathcal{PMA}_{\xi,n}\Big|_{t_i}$ for $i=0,\ldots n-1$ (in the equations we use ξ as a placeholder for either y or z). From Eqs. (12) and (13) note that when going from n to $n+1$, in order to construct $2d\mathcal{PMA}_{y,n+1}$ and $2d\mathcal{PMA}_{z,n+1}$ we can start from $2d\mathcal{PMA}_{y,n}$ and $2d\mathcal{PMA}_{z,n}$, shift existing controls by two indices ($c_i\to c_{i+2}$), add two new control qubits (c_0,c_1), shift existing targets by one index ($t_i\to t_{i+1}$), add a new target qubit t_0, and finally we add gates on t_0 according to Eqs. (12) and (13). Now lets analyze $2d\mathcal{MAR}_y(\boldsymbol{\Theta}_{n+1}^{(P,y)})$ and $2d\mathcal{MAR}_z(\boldsymbol{\Theta}_{n+1}^{(P,z)})$.

Keeping in mind that the goal is to show that $2d\mathcal{MAR}_\xi(\boldsymbol{\Theta}_{n+1}^{(P,\xi)})\Big|_{t_i}\cong 2d\mathcal{PMA}_{\xi,n+1}\Big|_{t_i}$ for $i=0,\ldots,n$, from Proposition 27 we have the following:

$$\hat{\boldsymbol{\Theta}}_{n+1}^{(P,z)}=\begin{bmatrix}\underbrace{\pi/2,0,\cdots,0}_{4^n} & \underbrace{0,\cdots,0,-\pi/2}_{4^n} & \underbrace{0,\cdots,0}_{4^n} & \underbrace{0,\cdots,0}_{4^n}\\ \hat{\boldsymbol{\Theta}}_n^{(P,z)} & 0 & 0 & 0\end{bmatrix},$$

$$\hat{\mathbf{\Theta}}_{n+1}^{(P,y)} = \begin{bmatrix} \underbrace{\pi/2, 0, \cdots, 0}_{4^n} & \underbrace{0, \cdots, 0}_{4^n} & \underbrace{-\pi/2, 0, \cdots, 0}_{4^n} & \underbrace{0, \cdots, 0}_{4^n} \\ \hat{\mathbf{\Theta}}_{n}^{(P,y)} & 0 & 0 & 0 \end{bmatrix}.$$

Each row in the above matrices correspond to a target qubit, which correspond to a row in the circuit $2d\mathcal{MXR}_\xi(\mathbf{\Theta}_{n+1}^{(P,\xi)})$. There are two types of rows.

1. "Old rows", i.e., rows with index $i \geq 1$. These rows already exist at level n. At level $n+1$ they are extended by trailing zeros. All angles beyond column 4^n are zero, and corresponding rotations are identities and theirs gates can be removed. Next, we argue that $2d\mathcal{MXR}_\xi(\mathbf{\Theta}_{n+1}^{(P,\xi)})\Big|_{t_i}$ equivalent to $2d\mathcal{MXR}_y(\mathbf{\Theta}_{n}^{(P,\xi)})|_{t_{i-1}}$ with shifted controls ($c_{i-2} \mapsto c_i$). To do so, we show that

$$2d\mathcal{MXR}_\xi(\mathbf{\Theta}_{n+1}^{(P,\xi)})\Big|_{t_i} \cong \mathcal{F}^{(2)}_{4^{n+1}}\left(\phi_{i,0}, \ldots, \phi_{i,4^n-1}\right) \cdot \mathrm{CX}(c_2, t_i), \tag{14}$$

for some angles $\phi_{i,0}, \ldots, \phi_{i,4^n-1}$ where $\mathcal{F}^{(2)}_{4^{+1}}(\cdot)$ denotes a row in the recursive multiplexer block from depth 2 onward, i.e., row in the circuit $2d\mathcal{MXR}_\xi(\mathbf{\Theta}_{n}^{(P,\xi)})$ with relabeled wires ($c_j \mapsto c_{j+2},\ t_{i-1} \mapsto t_i$). Then, for $i = 1, \ldots, n$

$$\mathcal{F}^{(2)}_{4^{n+1}}\left(\phi_{i,0}, \ldots, \phi_{i,4^n-1}\right) \cdot \mathrm{CX}(c_2, t_i) \cong 2d\mathcal{MXR}_\xi(\mathbf{\Theta}_{n}^{(P,\xi)})|_{t_{i-1}}$$

To see Eq. (14), expand the recursion for the block of size 4^{q+1} for any t_i, $i \geq 1$ we have:

$$\begin{aligned}
\mathcal{F}^{(0)}_{4^{n+1}}(\underbrace{\phi_{i,0}, \ldots, \phi_{i,4^n-1}}_{4^n \text{ zeros}}, \underbrace{0, 0, \ldots, 0}_{3\cdot 4^q \text{ zeros}}) &= \mathcal{F}^{(1)}_{4^{n+1}}(\underbrace{\phi_{i,0}, \ldots, \phi_{i,4^n-1}}_{4^n \text{ zeros}}, \underbrace{0, 0, \ldots, 0}_{4^n \text{ zeros}}) \cdot \mathrm{CX}(c_0, t_i) \cdot \mathcal{F}^{(1)}_{4^{n+1}}(\underbrace{0, 0, \ldots, 0}_{2\cdot 4^n \text{ zeros}}) \\
&\cong \mathcal{F}^{(2)}_{4^{n+1}}(\underbrace{\phi_{i,0}, \ldots, \phi_{i,4^n-1}}_{4^n \text{ zeros}}) \cdot \mathrm{CX}(c_1, t_i) \cdot \mathcal{F}^{(2)}_{4^{n+1}}(\underbrace{0, 0, \ldots, 0}_{4^n \text{ zeros}}) \cdot \mathrm{CX}(c_0, t_i) \cdot \mathrm{CX}(c_1, t_i) \\
&\cong \mathcal{F}^{(2)}_{4^{n+1}}(\underbrace{\phi_{i,0}, \ldots, \phi_{i,4^n-1}}_{4^n \text{ zeros}}) \cdot \mathrm{CX}(c_1, t_i) \cdot \mathrm{CX}(c_2, t_i) \cdot \mathrm{CX}(c_0, t_i) \cdot \mathrm{CX}(c_1, t_i) \\
&\cong \mathcal{F}^{(2)}_{4^{n+1}}(\underbrace{\phi_{i,0}, \ldots, \phi_{i,4^n-1}}_{4^q \text{ zeros}}) \cdot \mathrm{CX}(c_2, t_i) \cdot \mathrm{CX}(c_0, t_i)
\end{aligned}$$

combining with the full circuit the top level closing CNOT (the final circuit is $2d\mathcal{MXR}_\xi(\mathbf{\Theta}_{n+1}^{(P,\xi)})\Big|_{t_i} = \mathcal{F}^{(0)}_{4^{n+1}}(\hat{\boldsymbol{\theta}}) \cdot \mathrm{CX}(c_0, t_i)$ for some row $\hat{\boldsymbol{\theta}}$)

$$\begin{aligned}
\mathcal{F}^{(0)}_{4^{n+1}}(\underbrace{\phi_{i,0}, \ldots, \phi_{i,4^n-1}}_{4^n \text{ zeros}}, \underbrace{0, 0, \ldots, 0}_{3\cdot 4^n \text{ zeros}}) \cdot \mathrm{CX}(c_0, t_i) &\cong F^{(2)}(\underbrace{\phi_{i,0}, \ldots, \phi_{i,4^n-1}}_{4^n \text{ zeros}}) \cdot \mathrm{CX}(c_2, t_i) \cdot \mathrm{CX}(c_0, t_i) \cdot \mathrm{CX}(c_0, t_i) \\
&\cong \mathcal{F}^{(2)}_{4^{n+1}}\left(\phi_{i,0}, \ldots, \phi_{i,4^n-1}\right) \cdot \mathrm{CX}(c_2, t_i)
\end{aligned}$$

as desired. Now, use the inductive assumption that $2d\mathcal{MXR}_\xi(\mathbf{\Theta}_{n}^{(P,\xi)})|_{t_{i-1}} \cong 2d\mathcal{PMX}_{\xi,n}\Big|_{t_{i-1}}$ to show equivalence for "old rows".

2. A new top row with index $i = 0$. For $\hat{\mathbf{\Theta}}_{n+1}^{(P,z)}$ we have sparse row angles given in Proposition 27, and we need to show that

$$\begin{aligned}
2d\mathcal{MXR}_z(\mathbf{\Theta}_{n+1}^{(P,z)})\Big|_{t_0} &\cong \mathcal{R}_{z,t_0}\left(\pi/2\right) \cdot \mathrm{CX}(c_1, t_0) \cdot \mathcal{R}_{z,t_0}\left(-\pi/2\right) \cdot \mathrm{CX}(c_1, t_0) \\
2d\mathcal{MXR}_y(\mathbf{\Theta}_{n+1}^{(P,y)})\Big|_{t_0} &\cong \mathcal{R}_{y,t_0}\left(\pi/2\right) \cdot \mathrm{CX}(c_1, t_0) \cdot \mathrm{CX}(c_0, t_0) \cdot \mathcal{R}_{y,t_0}\left(-\pi/2\right) \cdot \mathrm{CX}(c_1, t_0) \cdot \mathrm{CX}(c_0, t_0)
\end{aligned}$$

Using the induction hypothesis for z, Eq. (2), Lemma 30 and Proposition 31, we have

$$\begin{aligned}\mathcal{F}^{(0)}_{4^{n+1}}(\underbrace{\pi/2,0,\ldots,0}_{4^n\text{ zeros}},\underbrace{0,0,\ldots,-\pi/2}_{4^n\text{ zeros}},\underbrace{0,0,\ldots,0}_{2\cdot 4^n\text{ zeros}}) &= \mathcal{F}^{(1)}_{4^{n+1}}(\underbrace{\pi/2,0,\ldots,0}_{4^n},\underbrace{0,\ldots 0,-\pi/2}_{4^n})\cdot \mathrm{CX}(c_0,t_0)\cdot \mathcal{F}^{(1)}_{4^{n+1}}(\underbrace{0,0,\ldots,0}_{2\cdot 4^n\text{ zeros}})\\ &\cong F^{(1)}(\underbrace{\pi/2,0,\ldots,0}_{4^n})\cdot \mathrm{CX}(c_1,t_0)\cdot F^{(1)}(\underbrace{0,\ldots 0,-\pi/2}_{4^n})\cdot \mathrm{CX}(c_0,t_0)\cdot \mathrm{CX}(c_1,t_0)\cdot\\ &\cong \mathcal{R}_{z,t_0}(\pi/2)\cdot \mathrm{CX}(c_1,t_0)\cdot \mathrm{CX}(c_1,t_0)\cdot \mathrm{CX}(c_1,t_0)\cdot \mathcal{R}_{z,t_0}(\pi/2)\cdot \mathrm{CX}(c_0,t_0)\cdot \mathrm{CX}(c_1,t_0)\\ &\cong \mathcal{R}_{z,t_0}(\pi/2)\cdot \mathrm{CX}(c_1,t_0)\cdot \mathcal{R}_{z,t_0}(\pi/2)\cdot \mathrm{CX}(c_0,t_0)\cdot \mathrm{CX}(c_1,t_0)\end{aligned}$$

combining with the full circuit the top level closing CNOT,

$$\mathcal{F}^{(0)}_{4^{n+1}}(\underbrace{\pi/2,0,\ldots,0}_{4^n\text{ zeros}},\underbrace{0,0,\ldots,-\pi/2}_{4^n\text{ zeros}},\underbrace{0,0,\ldots,0}_{2\cdot 4^n\text{ zeros}})\cdot \mathrm{CX}(c_0,t) = \mathcal{R}_{z,t_0}(\pi/2)\cdot \mathrm{CX}(c_1,t_0)\cdot \mathcal{R}_{z,t_0}(\pi/2)\cdot \mathrm{CX}(c_1,t_0)$$

The proof the y case follows the same logic. For $\hat{\mathbf{\Theta}}^{(P,y)}_{n+1}$, we expand

$$\mathcal{F}^{(0)}_{4^{n+1}}(\underbrace{\pi/2,0,\ldots,0}_{4^n\text{ zeros}},\underbrace{0,0,\ldots,0}_{4^n\text{ zeros}},\underbrace{-\pi/2,0,\ldots,0}_{4^n\text{ zeros}},\underbrace{0,0,\ldots,0}_{4^n\text{ zeros}})$$

as follows,

$$\begin{aligned}\mathcal{F}^{(0)}_{4^{n+1}}(\cdot) &= \mathcal{F}^{(1)}_{4^{n+1}}(\underbrace{\pi/2,0,\ldots,0}_{4^n\text{ zeros}},\underbrace{0,0,\ldots,0}_{4^n\text{ zeros}})\cdot \mathrm{CX}(c_0,t_0)\cdot \mathcal{F}^{(1)}_{4^{n+1}}(\underbrace{-\pi/2,0,\ldots,0}_{4^n\text{ zeros}},\underbrace{0,0,\ldots,0}_{4^n\text{ zeros}})\\ &\cong \mathcal{F}^{(2)}_{4^{n+1}}(\underbrace{\pi/2,0,\ldots,0}_{4^n\text{ zeros}})\cdot \mathrm{CX}(c_1,t_0)\cdot \mathcal{F}^{(2)}_{4^{n+1}}(\underbrace{0,0,\ldots,0}_{4^n\text{ zeros}})\cdot \mathrm{CX}(c_0,t_0)\cdot \mathcal{F}^{(2)}_{4^{n+1}}(\underbrace{-\pi/2,0,\ldots,0}_{4^n\text{ zeros}})\cdot \mathrm{CX}(c_1,t_0)\cdot \mathcal{F}^{(2)}_{4^{n+1}}(\underbrace{0,0,\ldots,0}_{4^n\text{ zeros}})\\ &\cong \mathcal{F}^{(2)}_{4^{n+1}}(\underbrace{\pi/2,0,\ldots,0}_{4^n\text{ zeros}})\cdot \mathrm{CX}(c_1,t_0)\cdot \mathrm{CX}(c_2,t_0)\cdot \mathrm{CX}(c_0,t_0)\cdot \mathcal{F}^{(2)}_{4^{n+1}}(\underbrace{-\pi/2,0,\ldots,0}_{4^n\text{ zeros}})\cdot \mathrm{CX}(c_1,t_0)\cdot \mathrm{CX}(c_2,t_0)\\ &\cong \mathcal{R}_{y,t_0}(\pi/2)\cdot \mathrm{CX}(c_2,t_0)\cdot \mathrm{CX}(c_1,t_0)\cdot \mathrm{CX}(c_2,t_0)\cdot \mathrm{CX}(c_0,t_0)\cdot \mathcal{R}_{y,t_0}(\pi/2)\cdot \mathrm{CX}(c_2,t_0)\cdot \mathrm{CX}(c_1,t_0)\cdot \mathrm{CX}(c_2,t_0)\\ &\cong \mathcal{R}_{y,t_0}(\pi/2)\cdot \mathrm{CX}(c_1,t_0)\cdot \mathrm{CX}(c_0,t_0)\cdot \mathcal{R}_{y,t_0}(\pi/2)\cdot \mathrm{CX}(c_1,t_0)\cdot\end{aligned}$$

combining with the full circuit the top level closing CNOT we have,

$$2d\mathcal{MXR}^{(n+1)}_y(\mathbf{\Theta}^{(y)})\Big|_{t_0} = \mathcal{R}_{y,t_0}(\pi/2)\cdot \mathrm{CX}(c_1,t_0)\cdot \mathrm{CX}(c_0,t_0)\cdot \mathcal{R}_{y,t_0}(\pi/2)\cdot \mathrm{CX}(c_1,t_0)\cdot \mathrm{CX}(c_0,t_0)$$

Combining the above, we have for any $i = 0,\ldots,n$ that

$$2d\mathcal{MXR}_\xi(\mathbf{\Theta}^{(P,\xi)}_{n+1})\Big|_{t_i} \cong 2d\mathcal{PMX}_{\xi,n+1}\Big|_{t_i}$$

as desired. □

5.2.3 Ultra low depth phase correction

Next, we consider how to do the phase correction required for implementing $\mathcal{PMX}_n$ via Eq. (9). Recall, the phases can be corrected by a diagonal gate for diagonal values

$$\underbrace{\mathbf{e}^{\mathrm{i}\varphi_0},\ldots,\mathbf{e}^{\mathrm{i}\varphi_0}}_{n\text{ times}},\underbrace{\mathbf{e}^{\mathrm{i}\varphi_1},\ldots,\mathbf{e}^{\mathrm{i}\varphi_1}}_{n\text{ times}},\ldots,\underbrace{\mathbf{e}^{\mathrm{i}\varphi_{4^n-1}},\ldots,\mathbf{e}^{\mathrm{i}\varphi_{4^n-1}}}_{n\text{ times}}$$

where φ_j is equal to $\pi/2$ times number of X,Y and Z in $\mathbf{w}_j$. Let $\mathcal{D}^*_{(q)} = (\mathcal{D}^*)^{\otimes n}\otimes\mathcal{I}_n$ where $\mathcal{D}^*$ is the circuit given in Figure 11 (a). Note that $\mathbf{M}(\mathcal{D}^*) = \mathbf{diag}(1,\mathrm{i},\mathrm{i},\mathrm{i})$. In Figure 11 (b) we have a circuit diagram of $\mathcal{D}^*_{(n)}$. We show that this diagonal gate corrects the phases.

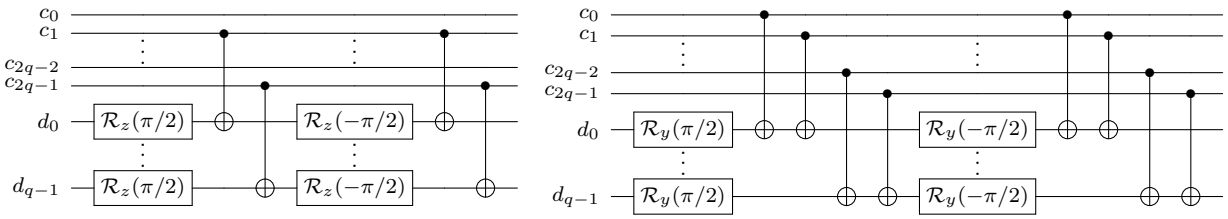


Figure 9: Quantum 2D rotation multiplexers for all Paulis. (a) Visualization of the $2d\mathcal{PMX}_{z,n}$. (b) Visualization of the $2d\mathcal{PMX}_{z,n}$.

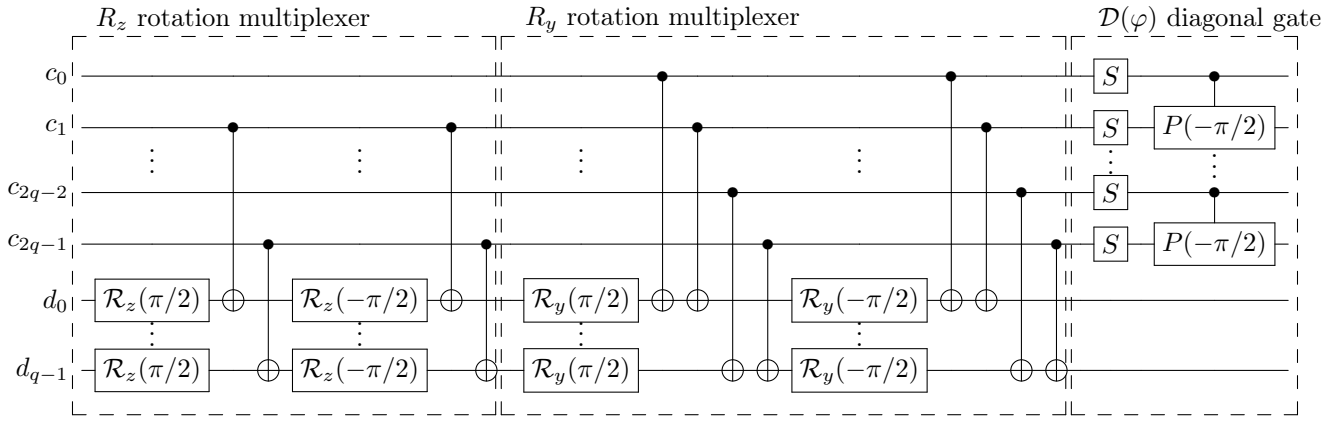


Figure 10: Block circuit diagram of the $\mathcal{PMX}_n$ implementation showing the three main components: (1) ultra-sparse $\mathcal{R}_z$ rotation multiplexer (2) ultra-sparse $\mathcal{R}_y$ rotation multiplexer , and (3) efficient diagonal gate $\mathcal{D}(\boldsymbol{\varphi})$ for phase correction. The dashed boxes highlight each functional block within the overall circuit architecture.

For each index j, we have $e^{\mathrm{i}\varphi_j} = \mathrm{i}^{n-c(j,q)}$ where $c(j,q)$ is the number of I letters in the Pauli word corresponding to index j. The key observation in the following recursion formula

$$c(j,n) = c(j', n-1) + \mathbf{1}_{r=0} \tag{15}$$

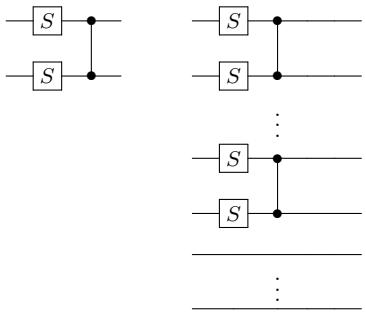
where $j = 4j' + r$ with $j' = \lfloor j/4 \rfloor$ and $r = j \bmod 4$, and $\mathbf{1}_{r=0} = 1$ if $r = 0$ and 0 otherwise. Therefore, $c(j,n) = c(\lfloor j/4 \rfloor, n-1) + \mathbf{1}_{j \bmod 4=0}$.

We now show by induction that

$$\begin{aligned} \mathbf{M}((\mathcal{D}^*)^{\otimes n}) &= \mathbf{diag}\left(\mathbf{e}^{\mathrm{i}\varphi_0}, \mathbf{e}^{\mathrm{i}\varphi_1}, \ldots, \mathbf{e}^{\mathrm{i}\varphi_{4^n-1}}\right) \\ &= \mathbf{diag}\left(\mathrm{i}^{n-c(0,n)}, \ldots, \mathrm{i}^{n-c(4^n-1,n)}\right) \end{aligned} \tag{16}$$

For $n = 1$, the claim follows since $c(0,1) = 1$ and $c(1,1) = c(2,1) = c(3,1) = 0$, this equals

$$\mathbf{diag}\left(\mathrm{i}^{1-c(0,1)}, \mathrm{i}^{1-c(1,1)}, \mathrm{i}^{1-c(2,1)}, \mathrm{i}^{1-c(3,1)}\right).$$



(a) (b)

Figure 11: (a) Circuit for $\mathcal{D}^* = \mathrm{diag}(1, \mathrm{i}, \mathrm{i}, \mathrm{i})$. (b) Circuit for $\mathcal{D}^*_{(n)} = (\mathcal{D}^*)^{\otimes n} \otimes \mathcal{I}_n$. It consists of n stacked copies of the two-qubit diagonal gate $\mathcal{D}^*$ acting on disjoint pairs of qubits, followed by n lower wires on which the identity acts.

As for $n > 1$, for any index $j \in \{0, 1, \ldots, 4^n - 1\}$, we can write $j = 4j' + r$ where $j' = \lfloor j/4 \rfloor$ and $r = j \bmod 4$. The (j, j)-th diagonal entry of $(\mathcal{D}^*)^{\otimes n}$ is:

$$(j', j')\text{-th entry of } (\mathcal{D}^*)^{\otimes n-1} \times (r, r)\text{-th entry of } \mathcal{D}^*$$

By the inductive hypothesis and the definition of $\mathcal{D}^*$, we find that the (j, j)-th diagonal entry of $\mathbf{M}((\mathcal{D}^*)^{\otimes n})$ is

$$\mathrm{i}^{n-c(j',n-1)-\mathbf{1}_{r=0}}$$

By Eq. (15), $c(j, n) = c(j', n-1) + \mathbf{1}_{r=0}$, therefore:

$$\mathrm{i}^{n-c(j',n-1)-\mathbf{1}_{r=0}} = \mathrm{i}^{n-c(j,n)}$$

and we conclude by induction that Eq. (16) holds.

Now, we have that

$$\begin{aligned}
\mathbf{M}(\mathcal{D}^*_{(n)}) \cdot \bigoplus_{j=0}^{4^n-1} e^{-\mathrm{i}\varphi_j} \sigma_{\mathbf{w}_j} &= (\mathbf{M}((\mathcal{D}^*)^{\otimes n}) \otimes \mathbf{I}_{2^n}) \bigoplus_{j=0}^{4^n-1} e^{-\mathrm{i}\varphi_j} \sigma_{\mathbf{w}_j} \\
&= \left(\bigoplus_{j=0}^{4^n-1} e^{\mathrm{i}\varphi_j} \mathbf{I}_{2^n} \right) \cdot \left(\bigoplus_{j=0}^{4^n-1} e^{-\mathrm{i}\varphi_j} \sigma_{\mathbf{w}_j} \right) \\
&= \bigoplus_{j=0}^{4^n-1} \sigma_{\mathbf{w}_j}
\end{aligned}$$

5.2.4 Bringing $\mathcal{PMX}_n$ together

We now combine the ingredients of the previous subsections. Subsection 5.2.2 introduced ultra low depth rotation multiplexers for the specific ultra sparse angles relevant for the all Pauli multiplexer (see Subsection 5.2.1). Subsection 5.2.3 described the diagonal correction circuit $\mathcal{D}^*_{(n)}$, ensuring the correct phases. Putting these pieces together on after the other, the full multiplexer for all Pauli matrices is

$$\mathcal{PMX}_n \coloneqq \mathcal{D}^*_{(n)} \cdot 2d\mathcal{PMX}_{y,n} \cdot 2d\mathcal{PMX}_{z,n}.$$

A block diagram appears in Figure 10.

Complexity of $\mathcal{PMX}_n$:

- **Gate count:** $4n$ rotation gates, $6n$ CNOTS, $2n$ S gates, and n controlled phase gate (non-Clifford gate, T-depth 1 and T-count 1 for phase $-\pi/2$).
- **Depth:** 10.
- **T-count:** all come from the controlled phase gates. There are n such gates, so T-count is n.
- **T-depth:** all controlled phase gates are in parallel, so T-depth is 1.

Remarkably, depth is constant regardless of how many matrices are multiplexed, and the number of gates is logarithmic in the number multiplexed Pauli matrices.

6 From dense classical matrices to block encodings circuits

This section presents algorithms for directly constructing block encoding circuits from classical matrix representations. Other types of conversions are discussed in the next section. We address both Hermitian and non-Hermitian cases and consider matrices specified in either the standard or Pauli basis. The core methodology is based on the Linear Combination of Unitaries (LCU) approach and on the building blocks introduced in the preceding sections.

6.1 LCU block encodings framework

A variety of approaches have been proposed in the literature for general purpose block encoding [32, 8], many of them based on the LCU approach. In this subsection, we review how LCUs can be used to construct block encoding.

To construct a block encoding via LCU a classical matrix $\mathbf{A} \in \mathbb{C}^{N\times N}$ is expressed as a weighted sum where each summand $\mathbf{U}_i \in \mathbb{C}^{N\times N}$ is a unitary operator and the weights are given by a $\boldsymbol{\nu} = (\nu_0, \ldots, \nu_{K-1})$ coefficient vector, where K is a power of 2, i.e.,

$$\mathbf{A} = \sum_{i=0}^{K-1} \nu_i \mathbf{U}_i, \quad \nu_i \in \mathbb{C}, \tag{17}$$

With such a decomposition, one can construct a block encoding as:

$$\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}} = \left(\mathcal{U}_{\sqrt{\boldsymbol{\nu}}}^{\mathrm{SP}} \otimes \mathcal{I}_n\right) \cdot \mathcal{MX}(\mathcal{U}_1, \ldots, \mathcal{U}_{K-1}) \cdot \left((\mathcal{U}_{\sqrt{\boldsymbol{\nu}}}^{\mathrm{SP}})^{\mathrm{T}} \otimes \mathcal{I}_n\right) \tag{18}$$

where $\mathcal{U}_j$ is a circuit that implements the unitary $\mathbf{U}_j$ (i.e., $\mathbf{M}(\mathcal{U}_j) = \mathbf{U}_j$), and $\mathcal{U}_{\sqrt{\boldsymbol{\nu}}}^{\mathrm{SP}} \in \mathbf{SP}_{\|\sqrt{\boldsymbol{\nu}}\|_F}\left(\sqrt{\boldsymbol{\nu}}\right)$ is a exact state preparation with $\sqrt{\boldsymbol{\nu}}$ denotes taking entry-wise square-root of $\boldsymbol{\nu}$. We have $\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}} \in \mathbf{BE}_{\|\sqrt{\boldsymbol{\nu}}\|_F^2}(\mathbf{A})$.

When $\mathbf{A}$ is Hermitian it is desirable to build a Hermitian block encoding instead of a regular block encoding. The construction in Eq. (18) will yield a Hermitian block encoding only if all coefficients are nonnegative real numbers ($\nu_i \in \mathbb{R}_{\geq 0}$). For general real coefficients (possibly negative, i.e., $\nu_i \in \mathbb{R}$), we can construct a Hermitian block encoding by incorporating sign information through a diagonal gate:

$$\mathcal{U}_{\mathbf{A}}^{\mathrm{HBE}} = \left(\mathcal{U}_{\sqrt{|\boldsymbol{\nu}|}}^{\mathrm{SP}} \otimes \mathcal{I}_n\right) \cdot \mathcal{MX}(\mathcal{U}_0, \ldots, \mathcal{U}_{K-1}) \cdot (\mathcal{D}(\boldsymbol{\psi}) \otimes \mathcal{I}_n) \cdot \left((\mathcal{U}_{\sqrt{|\boldsymbol{\nu}|}}^{\mathrm{SP}})^{*} \otimes \mathcal{I}_n\right) \tag{19}$$

where $\boldsymbol{\psi} = \mathrm{sign}(\boldsymbol{\nu})$ and $\mathcal{U}_{\sqrt{|\boldsymbol{\nu}|}}^{\mathrm{SP}} \in \mathbf{SP}_{\|\sqrt{|\boldsymbol{\nu}|}\|_F}\left(\sqrt{|\boldsymbol{\nu}|}\right)$is a exact state preparation with $\sqrt{|\boldsymbol{\nu}|}$ denotes taking entry-wise square-root of the absolute value of $\boldsymbol{\nu}$. We have $\mathcal{U}_{\mathbf{A}}^{\mathrm{HBE}} \in \mathbf{HBE}_{\|\boldsymbol{\nu}\|_F^2}(\mathbf{A})$. However, this construction involves additional complexity due to the diagonal gate and must be applied for any sequence of signs

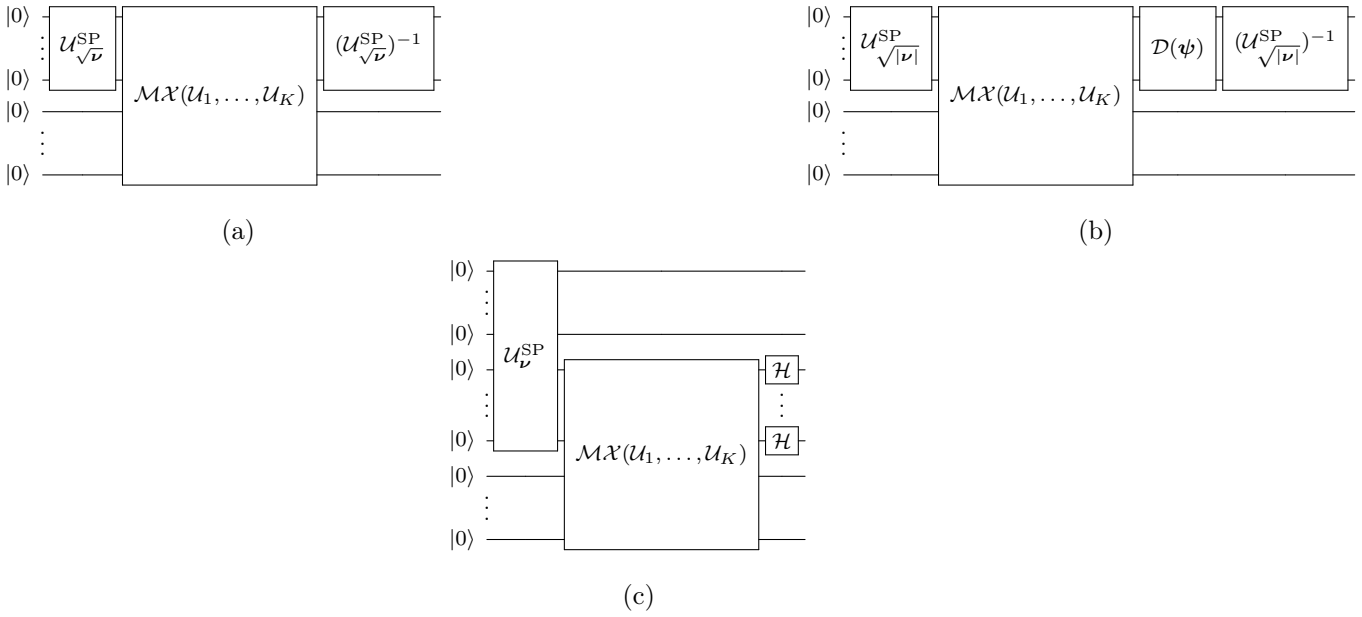


Figure 12: Linear Combination of Unitaries (LCU) block encoding constructions. (a) Standard LCU block encoding with complex coefficients using exact state preparation of $\sqrt{\boldsymbol{\nu}}$. (b) Hermitian block encoding (HBE) with real coefficients using additional diagonal gate $\mathcal{D}(\boldsymbol{\psi})$ for sign correction. (c) Alternative construction using full coefficient state preparation $\mathcal{U}^{\mathrm{SP}}_{\boldsymbol{\nu}}$ followed by Hadamard gates on $2k$ coefficients qubits. The circuits in (a) and (b) use k coefficients qubits (top) and n data qubits (bottom), while the circuit in (c) accommodates non-exact state preparation and uses p state preparation ancillary qubits (top), k coefficient qubits (middle) and n data qubits (bottom).

(arbitrary angels), as it depends on the input coefficients yielding a depth complexity of $O(K)$. Even if we have an highly efficient multiplexer and state preparation of $\sqrt{\boldsymbol{\nu}}$, the diagonal gate might be a bottleneck. We have $\mathcal{U}^{\mathrm{HBE}}_{\mathbf{A}} \in \mathbf{HBE}_{\|\boldsymbol{\nu}\|_1}(\mathbf{A})$.

Remark 33. While the LCU method is typically described in the literature with positive, real coefficients, it is straightforward to extend the decomposition to include complex coefficients [21, Lemma 52]. When complex coefficients are used, the inverse operation (conjugate transpose) typically employed for the positive, real case is replaced by the transpose (without the conjugate). This works because the transpose puts the amplitudes of $\sqrt{\boldsymbol{\nu}}$ in the first of the associated unitary matrix.

In scenarios where the coefficients are provided via a state preparation circuit $\mathcal{U}^{\mathrm{SP}}_{\boldsymbol{\nu}} \in \mathbf{SP}_{\alpha}(\boldsymbol{\nu})$ for some scale α with p ancillary qubits, rather than as a classical vector, an analogous construction can be used:

$$\mathcal{U}^{\mathrm{BE}}_{\mathbf{A}} = \left(\mathcal{U}^{\mathrm{SP}}_{\boldsymbol{\nu}} \otimes \mathcal{I}_n\right) \cdot \left(\mathcal{I}_p \otimes \mathcal{MX}(\mathcal{U}_0,\ldots,\mathcal{U}_{K-1})\right) \cdot \left(\mathcal{I}_p \otimes \mathcal{H}_k \otimes \mathcal{I}_n\right) \tag{20}$$

where $\mathcal{H}_k$ denotes applying $k = \log_2 K$ Hadamard gates to different qubits (we assume K is a power of 2). We have $\mathcal{U}^{\mathrm{BE}}_{\mathbf{A}} \in \mathbf{BE}_{\sqrt{2^k}\alpha}(\mathbf{A})$, so this construction has a worse scale factor.

Figure 12 presents the block diagram implementations for the three LCU block encoding approaches: (a) the standard construction with complex coefficients, (b) the Hermitian block encoding with sign correction, and (c) the alternative Hadamard-based construction.

Remark 34. We note that the constructions in Eqs. (19) and (18) cannot accommodate inexact state preparation with ancillary qubits, due to the fact that the "garbage" elements would be accounted for in the final

LCU, causing the unitary to fail to encode $\mathbf{A}$. However, the Hadamard based construction in Eq. (20) is designed to handle this, as the $\mathcal{I}_p \otimes \mathcal{H}_k$ operation prepares a state that effectively zero-pads the subspace where the garbage terms would appear. This ensures the garbage elements from the state preparation are multiplied by zero and are therefore eliminated from the final block-encoding.

6.2 From Pauli coefficient hypermatrix to block encoding

Block encodings of a matrix can be constructed using the matrix's Pauli decomposition [17]. Via our highly efficient multiplexer of all Pauli matrices (Section 5) we obtain an efficient block encoding. Consider a Pauli coefficient hypermatrix $\mathcal{A}_P$ associated with a matrix $\mathbf{A} \in \mathbb{C}^{N \times N}$. Our algorithms treat $\mathcal{A}_P$ as dense. A block encoding $\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}}$ can be constructed as follows:

$$\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}} = \left(\mathcal{U}^{\mathrm{SP}}_{\sqrt{\mathbf{vec}(\mathcal{A}_P)}} \otimes \mathcal{I}_{\log N}\right) \cdot \mathcal{PMX}_n \cdot \left((\mathcal{U}^{\mathrm{SP}}_{\sqrt{\mathbf{vec}(\mathcal{A}_P)}})^{\mathrm{T}} \otimes \mathcal{I}_{\log N}\right). \tag{21}$$

This construction works because $\mathbf{M}(\mathcal{PMX}_n)$ is a block diagonal matrix with all higher-order Pauli matrices as diagonal blocks, and since $\mathbf{A}$ is the weighted sum of these blocks.

When $\mathbf{A}$ is Hermitian, all the coefficients are real, and we can construct an Hermitian block encoding via:

$$\mathcal{U}_{\mathbf{A}}^{\mathrm{HBE}} = \left(\mathcal{U}^{\mathrm{SP}}_{\sqrt{|\mathbf{vec}(\mathcal{A}_P)|}} \otimes \mathcal{I}_{\log N}\right) \cdot \mathcal{PMX}_n \cdot (\mathcal{D}(\mathrm{sgn}(\mathbf{vec}\,(\mathcal{A}_P)) \otimes \mathcal{I}_{\log N}) \cdot \left((\mathcal{U}^{\mathrm{SP}}_{\sqrt{|\mathbf{vec}(\mathcal{A}_P)|}})^{*} \otimes \mathcal{I}_{\log N}\right), \tag{22}$$

The scale factor for both block encodings is $\|\mathbf{vec}\,(\mathcal{A}_P)\,\|_1$(see Remark 34). The classical cost of constructing the circuit is $O(N^2 \log N)$ due to the cost of constructing the state preparation component. The circuit has $3\log N$ qubits, and the depth is $2d\left(\mathcal{U}^{\mathrm{SP}}_{\sqrt{|\mathbf{vec}(\mathcal{A}_P)|}}\right) + d\left(\mathcal{D}(\mathrm{sgn}(\mathbf{vec}\,(\mathcal{A}_P)))\right) + 12 = O(N^2/n)$ due to the state preparation circuit. The T-cost is equal to $d_T(\mathcal{U}^{\mathrm{SP}}_{\sqrt{|\mathbf{vec}(\mathcal{A}_P)|}}) + d_T(\mathcal{D}(\mathrm{sgn}(\mathbf{vec}\,(\mathcal{A}_P)))) + 1 = O(\frac{N^2}{n}\log(N))$.[5]

Algorithm 2 provides a formal description for this block encoding construction based on Eqs. (21) and (22). Figure 13 illustrates the block encoding construction for the both cases.

We can also use Eq. (20) to build a block encoding:

$$\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}} = \left(\mathcal{U}^{\mathrm{SP}}_{\mathbf{vec}(\mathcal{A}_P)} \otimes \mathcal{I}_n\right) \cdot (\mathcal{I}_p \otimes \mathcal{PMX}_n) \cdot \left(\mathcal{I}_p \otimes \mathcal{H}_{2n} \otimes \mathcal{I}_n\right). \tag{23}$$

In the above, we can use *any* $\mathcal{U}^{\mathrm{SP}}_{\mathbf{vec}(\mathcal{A}_P)} \in \mathbf{SP}_\alpha\,(\mathbf{vec}\,(\mathcal{A}_P))$, and p denotes the number of ancillary qubits in $\mathcal{U}^{\mathrm{SP}}_{\mathbf{vec}(\mathcal{A}_P)}$ (so $\mathcal{U}^{\mathrm{SP}}_{\mathbf{vec}(\mathcal{A}_P)}$ has $2n+p$ qubits). The classical cost of constructing the circuit is $O(N^2 \log N)$ due to the cost of constructing the state preparation component. The circuit has $3n+p$ qubits, and the depth is $d(\mathcal{U}^{\mathrm{SP}}_{\mathbf{vec}(\mathcal{A}_P)}) + 12$. The T-cost is equal to the T-cost of the state preparation circuit plus one. See Figure 13 (c) for a block diagram.

The scale of $\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}}$ is $N\alpha$. When using a exact state preparation for $\mathbf{vec}\,(\mathcal{A}_P)$ we have $\alpha = \|\mathcal{A}_P\|_F$ and we get a scale factor of $\sqrt{N} \cdot \|\mathbf{A}\|_F$ which is always larger or equal to $\|\mathbf{vec}\,(\mathcal{A}_P)\,\|_1$.However, the construction in Eq. (23) has an advantage in that any state preparation for $\mathbf{vec}\,(\mathcal{A}_P)$ can be used, and this can allow us to trade qubits for depth (we see this in the depth calculation $d(\mathcal{U}^{\mathrm{SP}}_{\mathbf{vec}(\mathcal{A}_P)}) + 12$). Indeed, we can utilize various tradeoffs that have been explored in the literature between number of qubits and depth in state preparation circuits [54]. This can result in circuits that have lower depth at the cost of additional qubits. For example, using constructions similar to the ones described in [15], we can reduce depth to $O(\log(N))$ if the number of qubits increase to $O(N^2)$. We leave detailed exploration of these tradeoffs to future research.

[5] See Remark 4 for discussion on how is it possible for the T-depth to be larger than the regular depth.

Algorithm 2 PAULI2HBE: Create a Hermitian block encoding of Hermitian matrix $\mathbf{A}$ from Pauli coefficients.

Input: $\mathcal{A}_P$ of Hermitian matrix $\mathbf{A} \in \mathbb{H}_N$

$\boldsymbol{\psi} \leftarrow \text{sign}(\mathbf{vec}(\mathcal{A}_P))$
$\mathcal{D}(\boldsymbol{\psi}) \leftarrow \text{DiagonalGate}(\boldsymbol{\psi})$ {Construct diagonal gate, e.g. using Qiskit's `DiagonalGate`.}
Construct state preparation circuit $\mathcal{U}_{\sqrt{|\mathbf{vec}(\mathcal{A}_P)|}}$
$\mathcal{U} \leftarrow \left(\mathcal{U}_{\sqrt{|\mathbf{vec}(\mathcal{A}_P)|}} \otimes \mathcal{I}_{\log N}\right) \cdot \mathcal{PMX}_n \cdot (\mathcal{D}(\boldsymbol{\psi}) \otimes \mathcal{I}_{\log N}) \cdot \left(\mathcal{U}^*_{\sqrt{|\mathbf{vec}(\mathcal{A}_P)|}} \otimes \mathcal{I}_{\log N}\right)$
$\mathcal{U}_{\mathbf{A}}^{\text{HBE}} \leftarrow \text{BlockEncoding}(\text{U}=\mathcal{U}, \text{ shape}=N \times N, \text{ scale}=\|\mathbf{vec}(\mathcal{A}_P)\|_1)$
return $\mathcal{U}_{\mathbf{A}}^{\text{HBE}}$

6.3 Constructing a block encoding from a (dense) classical matrix

Suppose we are given the matrix $\mathbf{A}$ explicitly as entries in classical memory (in the standard basis). A block encoding can be constructed by converting the matrix to the Pauli basis using Algorithm 1 (DENSE2PAULI), and then proceeding as in the previous section. We call this algorithm MATRIX2BE; see Algorithm 3 for a pseudo-code description. The classical cost of constructing the circuit is $O(N^2 n)$, due to the cost of computing the Pauli coefficients and of constructing their exact state preparation (same asymptotic cost for both). A variant of the algorithm, called MATRIX2HBE, constructs an Hermitian block encoding (for Hermitian matrices), with similar costs; Algorithm 3 also specifies this variant. Given such a classical $\mathbf{A}$, MATRIX2BE constructs a $\mathcal{U}_{\mathbf{A}}^{\text{BE}} \in \mathbf{BE}_{\|\mathbf{vec}(\mathcal{A}_P)\|_1}(\mathbf{A})$, and MATRIX2HBE constructs a $\mathcal{U}_{\mathbf{A}}^{\text{BE}} \in \mathbf{HBE}_{\|\mathbf{vec}(\mathcal{A}_P)\|_1}(\mathbf{A})$.

For both MATRIX2BE and MATRIX2HBE, the number of qubits in the resulting circuit is $3n$, and the depth is $O(N^2/n)$. To see that this is the depth for the Hermitian case, from the circuit description (Eq. (22)), we see the cost is $2d(\mathcal{U}_{\mathcal{A}_P}^{\text{SP}}) + d(\mathcal{D}(\boldsymbol{\psi})) + 10$ where 10 is due to the depth of $\mathcal{PMX}_n$. The depth of $\mathcal{U}_{\mathcal{A}_P}^{\text{SP}}$ is $O(N^2/n)$, while the diagonal gate can be built with depth $O(N/n)$, so the overall depth is $O(N^2/n)$. A similar analysis applies to the non-Hermitian circuit. Asymptotically, the T-depth is also the same as the T-depth of the state preparation circuit, which is $O(N^2)$.

Other methods for constructing a block-encoding for a dense classical matrix. There is scarce literature on constructing block encodings given a dense classical matrix $\mathbf{A}$ (most works focused on the sparse access model [1, 14]). Table 1 lists various algorithms, and compares them to MATRIX2BE in terms of several parameters.

The FABLE algorithm [11] was the first work to describe an algorithm for this tasks. It uses a multiplexer structure that results in a circuit depth of $O(N^2)$ and requires $2n + 1$ qubits. It's scaling factor is $N \max_{i,j} |a_{ij}|$where the N component results from the use of two Hadamard layers. Clader et al. [15] studied various tradeoffs for building block encoding using a general formula from [21] which was previously used only for constructing block encodings in the sparse access model. They offer two algorithms, one with minimal T-depth and the other with minimal T-count. The construction is based on circuits implementing QRAM. While all their methods result in circuits with a scaling factor of $\|\mathbf{A}\|_F$, along with excellent T-complexities for the target complexity, all his algorithms result in circuits with $\Omega(N)$ qubits (compared to $O(n)$ for all other algorithms, i.e., an exponential gap). Li et al. [35] introduced BITBLE, method for constructing block encoding quantum circuits with scaling factor of $\|\mathbf{A}\|_F$ and circuit depth $O(N^2)$ [35]. Our method achieves a superior circuit depth compared to other $O(n)$-qubit algorithms like FABLE and BITBLE: $O(N^2/n)$ versus $O(N^2)$ As for the scale factor, we have

$$\frac{\|\mathbf{A}\|_F}{\sqrt{N}} \leq \|\mathbf{vec}(\mathcal{A}_P)\|_1 \leq \sqrt{N} \cdot \|\mathbf{A}\|_F \tag{24}$$

where the lower bound is realized with, for example, the identity matrix, and the upper bound is realized,

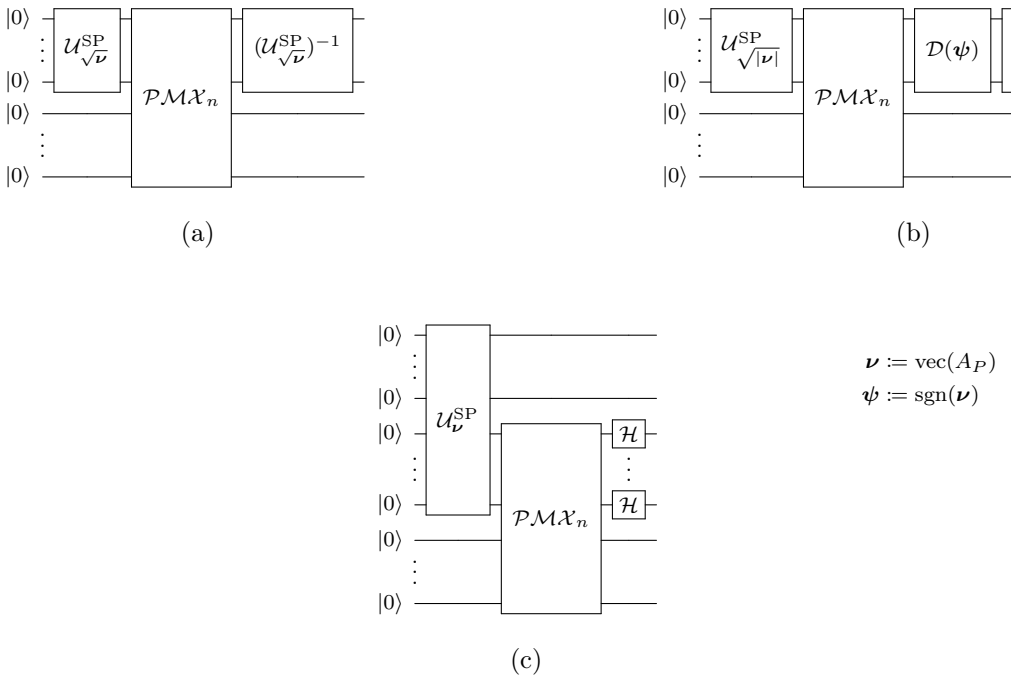


Figure 13: Pauli-based block encoding constructions. (a) Standard block encoding with complex coefficients using state preparation of $\sqrt{\mathbf{vec}\,(\mathcal{A}_P)}$. (b) Hermitian block encoding (HBE) with real coefficients using additional diagonal gate $\mathcal{D}(\mathrm{sgn}(\mathbf{vec}\,(\mathcal{A}_P))$ for sign correction. (c) Alternative construction using full coefficient state preparation $\mathcal{U}^{\mathrm{SP}}_{\mathbf{vec}(\mathcal{A}_P)}$ followed by Hadamard gates on $2n$ qubits. The circuits in (a) and (b) use $2n$ coefficients qubits (top) and n data qubits (bottom), while the circuit in (c) accommodates non-exact state preparation and uses p state preparation ancillary qubits (top), $2n$ coefficient qubits (middle) and n data qubits (bottom).

for example, when $\mathcal{A}_P$ is the all-ones tensor. Eq. (24) shows that our method obtains a scale factor that is not worse by factor more than $\sqrt{N}$ than the algorithms in [11, 15, 35], but may also be better by at most the same factor. Both [15, 35] also consider variants of their algorithm whose scaling factor is the so-called q-norm of a matrix. An interesting question is what is the minimal scale factor achievable with a generic block encoding methods for classical dense matrices.

7 Algorithms for converting between representations

Consider a dense matrix $\mathbf{A} \in \mathbb{C}^{N\times N}$ with no additional structure. Extending beyond the scope of the previous section, we identify five possible representations of $\mathbf{A}$: two classical and three based on quantum circuits[6]:

1. The entries of $\mathbf{A}$ in classical memory (in, e.g., row-major ordering).
2. The entries of $\mathcal{A}_P$ in classical memory (e.g., $\mathbf{vec}\,(\mathcal{A}_P)$).
3. As a matrix state preparation circuit of $\mathbf{A}$: $\mathcal{U}^{\mathrm{SP}}_{\mathbf{A}} \in \mathbf{MS}_\alpha\,(\mathbf{A})$ for some α.
4. As a hypermatrix state preparation circuit of $\mathcal{A}_P$: $\mathcal{U}^{\mathrm{SP}}_{\mathcal{A}_P} \in \mathbf{HS}_\alpha\,(\mathcal{A}_P)$ for some α.

[6]Representing in a quantum circuit is technically also classical: it is a classical description of a quantum circuit.

Algorithm 3 Matrix2BE: Create a block encoding from a classical matrix

1: **Input:** $\mathbf{A} \in \mathbb{C}^{N \times N}$

2: $\mathcal{A}_P \leftarrow$ DENSE2PAULI($\mathbf{A}$)
3: $\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}} \leftarrow$ PAULI2BE($\mathcal{A}_P$) (for Hermitian: $\mathcal{U}_{\mathbf{A}}^{\mathrm{HBE}} \leftarrow$ PAULI2HBE($\mathcal{A}_P$))
4: **return** $\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}}$ (for Hermitian: $\mathcal{U}_{\mathbf{A}}^{\mathrm{HBE}}$)

Algorithm	**#qubits**	**Scale factor**	**Depth**	**T-Depth**	**T-Count**	**Classical Complexity**
FABLE [11]	$2n+1$	$N \max_{i,j} \lvert a_{ij} \rvert$	$O(N^2)$	$O(nN^2)$		$O(N^2 n)$
Minimal depth [15]	$O(N^2)$	$\lVert \mathbf{A} \rVert_F$		$O(n)$	$O(N^2)$	
Minimal count [15]	$O(N)$	$\lVert \mathbf{A} \rVert_F$		$O(N)$	$O(N)$	
BITBLE Algorithm [35]	$2n$	$\lVert \mathbf{A} \rVert_F$	$O(N^2)$	$O(nN^2)$		$O(N^2 n)$
MATRIX2BE (THIS WORK)	$3n$	$\lVert \mathbf{vec}\,(\mathcal{A}_P) \rVert_1$	$O(N^2/n)$	$O(N^2)$	$O(nN^2)$	$O(N^2 n)$

Table 1: Comparison of of various algorithms for building a block encoding given dense $N \times N$ matrix in classical memory. Empty entires are ones in which the corresponding paper does not specify this complexity, and is not easily inferable from the description.

5. As block encoding circuit of $\mathbf{A}$: $\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}} \in \mathbf{BE}_{\alpha}\,(\mathbf{A})$ for some α.

In this section, we discuss algorithms for moving between representations, both from classical to quantum and from quantum to quantum (of different types).

In the previous sections, we developed the main tools for encoding matrices in quantum circuits, and introduced several key algorithms for specific settings. Since much of the heavy lifting has already been done, the descriptions in this section are concise and refer to the methodologies established earlier in the paper. Table 2 summarizes the algorithms, comparing their inputs, outputs, ancillary qubit requirements, gate complexity, scaling factor, and T-costs. Figure 14 also visually summarizes the various representations, and algorithm for converting between them. All algorithms have classical inputs and outputs, producing quantum circuit descriptions that can be executed on quantum hardware. Some conversions are already well-established in the literature, and our goal is simply to summarize the results and describe their computational complexity as part of a comprehensive description of algorithms to convert between representations. Other conversions, in particular the conversion between dense classical to block encoding and the conversion between state preparation to block encoding, are novel.

7.1 From classical to matrix state preparation

7.1.1 From coefficients in the standard basis

The conversion from a classical matrix representation in the standard basis to a matrix state preparation circuit is a well-established procedure in quantum computing literature [44, 51, 46]. Given a classical matrix $\mathbf{A} \in \mathbb{C}^{N \times N}$, we can construct a matrix state preparation circuit $\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}} \in \mathbf{MS}_{\lVert \mathbf{A} \rVert_F}\,(\mathbf{A})$ by vectorizing the matrix according to our definition in Section 2, and using standard state preparation techniques to create a circuit that prepares $||\mathbf{A}\rangle\rangle$.

Seminal works that describe the process of synthesizing arbitrary state preparation circuits are described in [44, 51]. These methods are based on rotation multiplexers (see Section 3), require $O(N^2 \log N)$ classical cost, produce circuits of depth and T-depth of $O(N^2)$, and use $O(N^2)$ gates. Various improvements were described over the years. The isometry method of [28] improves constant factors. Plesch and Brukner [46] showed how to reduce costs, specifically by improving the constant factors in the CNOT gate count and

Algorithm	Input	Output	Qubits	Depth	T-Depth	Subsec.
[54]	$\mathbf{A} \in \mathbb{C}^{N\times N}$	$\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}} \in \mathbf{MS}_{\|\mathbf{A}\|_F}(\mathbf{A})$	$2n$	$O(N^2/n)$	$O(N^2)$	7.1.1
MATRIX2PAULIMSP	$\mathbf{A} \in \mathbb{C}^{N\times N}$	$\mathcal{U}_{\mathcal{A}_P}^{\mathrm{SP}} \in \mathbf{HS}_{\|\mathcal{A}_P\|_F}(\mathcal{A}_P)$	$2n$	$O(N^2/n)$	$O(N^2)$	7.1.1
MATRIX2BE	$\mathbf{A} \in \mathbb{C}^{N\times N}$	$\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}} \in \mathbf{BE}_{\|\mathbf{vec}(\mathcal{A}_P)\|_1}(\mathbf{A})$	$3n$	$O(N^2/n)$	$O(N^2)$	6.3
MATRIX2HBE	$\mathbf{A} \in \mathbb{C}^{N\times N}$, Hermitian	$\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}} \in \mathbf{HBE}_{\|\mathbf{vec}(\mathcal{A}_P)\|_1}(\mathbf{A})$	$3n$	$O(N^2/n)$	$O(N^2)$	6.3
PAULIMATRIX2MSP	$\mathcal{A}_P \in \mathbb{C}^{4\times 4\times\cdots\times 4}$	$\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}} \in \mathbf{MS}_{\|\mathbf{A}\|_F}(\mathbf{A})$	$2n$	$O(N^2/n)$	$O(N^2)$	7.1.2
PAULIMATRIX2PAULIMSP	$\mathcal{A}_P \in \mathbb{C}^{4\times 4\times\cdots\times 4}$	$\mathcal{U}_{\mathcal{A}_P}^{\mathrm{SP}} \in \mathbf{HS}_{\|\mathcal{A}_P\|_F}(\mathcal{A}_P)$	$2n$	$O(N^2/n)$	$O(N^2)$	7.1.2
PAULIMATRIX2BE	$\mathcal{A}_P \in \mathbb{C}^{4\times 4\times\cdots\times 4}$	$\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}} \in \mathbf{BE}_{\|\mathbf{vec}(\mathcal{A}_P)\|_1}(\mathbf{A})$	$3n$	$O(N^2/n)$	$O(N^2)$	6.2
PAULIMSP2MSP	$\mathcal{U}_{\mathcal{A}_P}^{\mathrm{SP}} \in \mathbf{MS}_{\alpha}(\mathcal{A}_P)$	$\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}} \in \mathbf{MS}_{\sqrt{N}\alpha}(\mathbf{A})$	$+0$	$+6$	$+0$	4.3
PAULIMSP2BE	$\mathcal{U}_{\mathcal{A}_P}^{\mathrm{SP}} \in \mathbf{MS}_{\alpha}(\mathcal{A}_P)$	$\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}} \in \mathbf{BE}_{N\alpha}(\mathbf{A})$	$+n$	$+12$	$+1$	7.2.2
MSP2PAULIMSP	$\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}} \in \mathbf{MS}_{\alpha}(\mathbf{A})$	$\mathcal{U}_{\mathcal{A}_P}^{\mathrm{SP}} \in \mathbf{HS}_{\alpha/\sqrt{N}}(\mathcal{A}_P)$	$+0$	$+6$	$+0$	4.3
MSP2BE	$\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}} \in \mathbf{MS}_{\alpha}(\mathbf{A})$	$\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}} \in \mathbf{BE}_{N\alpha}(\mathbf{A})$	$+n$	$+18$	$+1$	7.2.2
BE2MSP	$\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}} \in \mathbf{BE}_{\alpha}(\mathbf{A})$	$\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}} \in \mathbf{MS}_{\sqrt{N}\alpha}(\mathbf{A})$	$+q_N$	$+2$	$+0$	7.2.1
BE2PAULIMSP	$\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}} \in \mathbf{BE}_{\alpha}(\mathbf{A})$	$\mathcal{U}_{\mathcal{A}_P}^{\mathrm{SP}} \in \mathbf{HS}_{\alpha}(\mathcal{A}_P)$	$+q_N$	$+8$	$+0$	7.2.1

Table 2: Summary of encoding algorithms and their resource requirements. For each algorithm, we list the input, output, number of *ancillary* qubits, circuit depth, T-depth, and the subsection where the construction is described. When the input is a circuit (bottom half of the table), columns "Qubits", "Depth" and "T-Depth" refer to *surplus* qubits or depth, i.e., qubits/depth on top of the qubit/depth of the input circuit. In this cases, we prefix with a plus symbol. We denote $q_N \coloneqq q(\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}}) - \log N$.

circuit depth for arbitrary state preparation using universal gate decompositions. Zhang et al. showed how depth and qubit count can be traded [60]. They show that $O(\log N)$ depth suffice if we allow the number of ancilla qubits to be $O(N)$. They also discuss preparing sparse vectors with D non zeroes entries, and prove near optimality of their costs (their proposed algorithm for sparse states achieves an exponentially faster depth of $O(\log ND)$ with $O(ND\log D)$ ancillary qubits). More recently, Sun et al. [54] proved an asymptotically optimal bound for general state preparation without ancillary qubits, achieving circuit depth $O(N^2/n)$ and T-depth [39, Table 1] of $O(N^2)$.

The conversion from a classical matrix representation in the standard basis to a hypermatrix state preparation circuit of $\mathcal{A}_P$ can be achieved through two methods. The straightforward way is to classically compute all coefficients of the Pauli hypermatrix $\mathcal{A}_P$ directly from the classical matrix $\mathbf{A}$, and then create the hypermatrix state preparation circuit $\mathcal{U}_{\mathcal{A}_P}^{\mathrm{SP}}$ using the same procedure described above. (This is essentially the same procedure as standard state preparation techniques once the vectorized form of the hypermatrix is obtained.) The second method is to build a matrix state preparation circuit for $\mathbf{A}$, and then use the change of basis techniques that we described in Subsection 4.3. Both methods have the same asymptotic cost (both classical and quantum).

7.1.2 From coefficients in the Pauli basis

Given a hypermatrix $\mathcal{A}_P$ representing the Pauli decomposition of matrix $\mathbf{A}$, we can create the matrix state preparation circuit $\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}}$ through two approaches. The first approach reconstructs the standard basis matrix $\mathbf{A}$ from $\mathcal{A}_P$ using the recursive Pauli decomposition relationship in Eq. (4), and then applies the standard matrix state preparation techniques described in the previous subsection. The second approach directly constructs the hypermatrix state preparation circuit $\mathcal{U}_{\mathcal{A}_P}^{\mathrm{SP}}$ and then applies a change of basis transformation to obtain $\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}}$. The change of basis from $\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}}$ to $\mathcal{U}_{\mathcal{A}_P}^{\mathrm{SP}}$ is described in Subsection 4.3. Both methods have the same asymptotic cost (both classical and quantum).

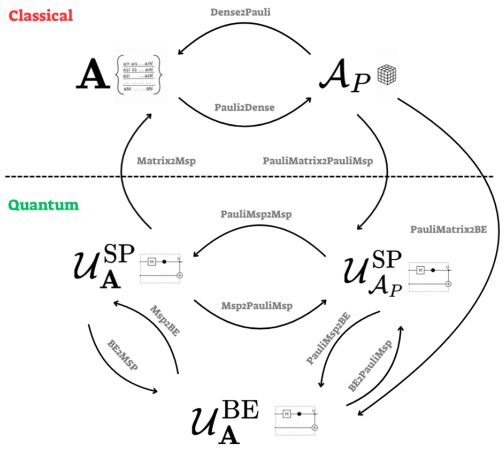


Figure 14: Illustration of the paper's algorithms. Top: classical representations $\mathbf{A}$ and $\mathcal{A}_P$. Bottom: quantum circuit representations ($\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}}$, $\mathcal{U}_{\mathcal{A}_P}^{\mathrm{SP}}$, and $\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}}$)

7.2 The interplay between matrix state preparation and block encoding

In this subsection, we explore the relationship between the two circuit representations of matrices discussed in this paper: matrix state preparation circuits and block encodings. A fundamental question is whether these two representations are equivalent, where we define equivalence as the ability to efficiently construct an efficient circuit for one representation given a circuit for the other representation with (roughly) the same complexities.

The conversion from a block encoding representation to a matrix state preparation representation was partially discussed in our previous work [42, Section 3.2.2]. It was shown that an exact block encoding can be converted to a matrix state preparation by attaching to it a very simple circuit. Here we complete the discussion, and show how convert from an arbitrary block encoding to a matrix state preparation. We also discuss the opposite conversion, from a state preparation to a block encoding. This conversion is much less straightforward, and requires a more elaborate circuit.

A surprising aspect of the results in this section is that an efficient bidirectional conversion between the two representations is possible. In particular, the conversion requires a surplus depth of $O(1)$, and the surplus T-depth is exactly 1. These costs are typically much lower than the ones required for the circuit themselves. However, the conversions do increase the number of qubits and worsen the scale of the representation.

We highlight one interesting consequence of the efficient bidirectional conversion. A fundamental tool in many quantum numerical linear algebra is the Quantum Singular Value Transformation (QSVT) algorithm [21]. However, this algorithm has only been defined for block encodings. Due to the results of this section, it can also be applied to matrix state preparation circuits.

7.2.1 From block encoding to matrix state preparation

In this subsection, the goal is to construct $\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}}$ and $\mathcal{U}_{\mathcal{A}_P}^{\mathrm{SP}}$ starting from $\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}}$. Due to the construction $\mathcal{U}_{\mathcal{A}_P}^{\mathrm{SP}} = \mathcal{U}^{(q)} \cdot \mathcal{U}_{\mathbf{A}}^{\mathrm{SP}}$ in Subsection 4.3, it suffices to establish the construction of $\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}}$ from $\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}}$.

Let $\mathbf{A} \in \mathbb{C}^{N \times N}$, and suppose we are given a block encoding circuit $\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}} \in \mathbf{BE}_{\alpha}(\mathbf{A})$ with $q(\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}})$ qubits. We show that we can construct $\mathcal{U}_{\mathbf{A}}^{\mathrm{SP}} \in \mathbf{MS}_{\sqrt{N}\alpha}(\mathbf{A})$ with $q(\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}}) + q_N$ qubits, where $q_N = q(\mathcal{U}_{\mathbf{A}}^{\mathrm{BE}}) - \log_2 N$ is the number of ancillary qubits in the block encoding. The key idea is to use an extended version of $\mathcal{U}_{\mathbf{I}_N}^{\mathrm{SP}}$ which is obtained by padding it with zero rows (see our previous work [42] for how to build $\mathcal{U}_{\mathbf{I}_N}^{\mathrm{SP}}$ and pad it

with zero rows). Let $\mathcal{U}^{\mathrm{SP}}_{\hat{\mathbf{I}}_N}$ be a matrix state preparation for

$$\hat{\mathbf{I}}_N = \begin{bmatrix} \mathbf{I}_N \\ \mathbf{0}_{(2^{q_N}-1)N} \end{bmatrix}.$$

Then,

$$\begin{aligned}
\mathcal{U}^{\mathrm{BE}}_{\mathbf{A}} \cdot_1 \mathcal{U}^{\mathrm{SP}}_{\hat{\mathbf{I}}_N} |0\rangle_{q(\mathcal{U}^{\mathrm{BE}}_{\mathbf{A}})+q_N} &= \mathcal{U}^{\mathrm{BE}}_{\mathbf{A}} \cdot_1 \left|\left| \begin{bmatrix} \frac{1}{\sqrt{N}}\mathbf{I}_N \\ \mathbf{0}_{(2^{q_N}-1)N} \end{bmatrix} \right\rangle\right\rangle \\
&= \left|\left| \mathbf{M}(\mathcal{U}^{\mathrm{BE}}_{\mathbf{A}}) \begin{bmatrix} \frac{1}{\sqrt{N}}\mathbf{I}_N \\ \mathbf{0}_{(2^{q_N}-1)N} \end{bmatrix} \right\rangle\right\rangle \\
&= \left|\left| \begin{bmatrix} \alpha^{-1}\mathbf{A} & * \\ * & * \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{N}}\mathbf{I}_N \\ \mathbf{0}_{(2^{q_N}-1)N} \end{bmatrix} \right\rangle\right\rangle \\
&= \left|\left| \begin{bmatrix} (\sqrt{N}\alpha)^{-1}\mathbf{A} \\ * \end{bmatrix} \right\rangle\right\rangle
\end{aligned}$$

and we have $\mathcal{U}^{\mathrm{SP}}_{\mathbf{A}} \in \mathbf{MS}_{\sqrt{N}\alpha}(\mathbf{A})$, i.e., a state preparation for $\mathbf{A}$ with $q(\mathcal{U}^{\mathrm{BE}}_{\mathbf{A}}) + q_N$ qubits.

7.2.2 From matrix state preparation to block encoding

In this subsection, the goal is to construct $\mathcal{U}^{\mathrm{BE}}_{\mathbf{A}}$ starting from $\mathcal{U}^{\mathrm{SP}}_{\mathbf{A}}$ or $\mathcal{U}^{\mathrm{SP}}_{\mathcal{A}_P}$. Again, due to the construction $\mathcal{U}^{\mathrm{SP}}_{\mathcal{A}_P} = \mathcal{U}^{(q)}\mathcal{U}^{\mathrm{SP}}_{\mathbf{A}}$ in Subsection 4.3, it suffices to establish the conversion from $\mathcal{U}^{\mathrm{SP}}_{\mathcal{A}_P}$ to $\mathcal{U}^{\mathrm{BE}}_{\mathbf{A}}$.

In Section 6.2 we show that given a classical Pauli coefficient hypermatrix we can construct block encoding. If we inspect the method, we see in Eq. (23) that the final circuit uses a state preparation circuit for $\mathcal{A}_P$. Thus, if we are given a state preparation for $\mathcal{A}_P$ we can just proceed to building the block encoding circuit. The final circuit is:

$$\mathcal{U}^{\mathrm{BE}}_{\mathbf{A}} = \left(\mathcal{U}^{\mathrm{SP}}_{\mathcal{A}_P} \otimes \mathcal{I}_{\log N}\right) \cdot \left(\mathcal{I}_p \otimes \mathcal{PMX}_N\right) \cdot \left(\mathcal{I}_p \otimes \mathcal{H}_{2\log N} \otimes \mathcal{I}_{\log N}\right)$$

In the case where $\mathcal{U}^{\mathrm{SP}}_{\mathcal{A}_P} \in \mathbf{HS}_\alpha(\mathcal{A}_P)$, the scale of $\mathcal{U}^{\mathrm{BE}}_{\mathbf{A}}$ is $N\alpha$ ($\sqrt{N}\|\mathbf{A}\|_F$ for exact state preparation). The classical cost of constructing the circuit is $O(g(\mathcal{U}^{\mathrm{SP}}_{\mathcal{A}_P}))$ dominated by the classical description of the state preparation component. The resulting circuit acts on $3n+p$ qubits, has depth of $d(\mathcal{U}^{\mathrm{SP}}_{\mathcal{A}_P})+12$, and its T-cost equals the T-cost of the state preparation plus one.

Acknowledgments.

This research was supported by the US-Israel Binational Science Foundation (Grant no. 2017698), Israel Science Foundation (Grant no. 1524/23) and IBM Faculty Award. Liron Mor-Yosef acknowledges support by the Milner Foundation and the Israel Council for Higher Education.

References

[1] Andris Ambainis. Variable time amplitude amplification and quantum algorithms for linear algebra problems. In *STACS'12 (29th Symposium on Theoretical Aspects of Computer Science)*, volume 14, pages 636–647. LIPIcs, 2012.

[2] Matthew Amy, Dmitri Maslov, and Michele Mosca. Polynomial-time T-depth optimization of Clifford+T circuits via matroid partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(10):1476–1489, 2014.

[3] Dong An and Lin Lin. Quantum linear system solver based on time-optimal adiabatic quantum computing and quantum approximate optimization algorithm. *ACM Transactions on Quantum Computing*, 3(2):1–28, 2022.

[4] Israel F Araujo, Daniel K Park, Teresa B Ludermir, Wilson R Oliveira, Francesco Petruccione, and Adenilton J Da Silva. Configurable sublinear circuits for quantum state preparation. *Quantum Information Processing*, 22(2):123, 2023.

[5] Israel F Araujo, Daniel K Park, Francesco Petruccione, and Adenilton J da Silva. A divide-and-conquer algorithm for quantum state preparation. *Scientific Reports*, 11(1):6329, 2021.

[6] Ville Bergholm, Juha J Vartiainen, Mikko Mottonen, and Martti M Salomaa. Quantum circuits with uniformly controlled one-qubit gates. *Physical Review A*, 71(5):052330, 2005.

[7] Dennis S. Bernstein. *Matrix Mathematics: Theory, Facts, and Formulas*. Princeton University Press, second edition, 2011.

[8] Dominic W Berry, Andrew M Childs, and Robin Kothari. Hamiltonian simulation with nearly optimal dependence on all parameters. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 792–809. IEEE, 2015.

[9] Stephen S. Bullock and Igor L. Markov. Asymptotically optimal circuits for arbitrary n-qubit diagonal comutations. *Quantum Info. Comput.*, 4(1):27–47, January 2004.

[10] Daan Camps, Lin Lin, Roel Van Beeumen, and Chao Yang. Explicit quantum circuits for block encodings of certain sparse matrices. *SIAM Journal on Matrix Analysis and Applications*, 45(1):801–827, 2024.

[11] Daan Camps and Roel Van Beeumen. FABLE: Fast approximate quantum circuits for block-encodings. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 104–113. IEEE, 2022.

[12] Shantanav Chakraborty, András Gilyén, and Stacey Jeffery. The Power of Block-Encoded Matrix Powers: Improved Regression Techniques via Faster Hamiltonian Simulation. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[13] Andrew M Childs and Robin Kothari. Simulating sparse Hamiltonians with star decompositions. In *Theory of Quantum Computation, Communication, and Cryptography: 5th Conference, TQC 2010, Leeds, UK, April 13-15, 2010, Revised Selected Papers 5*, pages 94–103. Springer, 2011.

[14] Andrew M Childs, Robin Kothari, and Rolando D Somma. Quantum algorithm for systems of linear equations with exponentially improved dependence on precision. *SIAM Journal on Computing*, 46(6):1920–1950, 2017.

[15] B David Clader, Alexander M Dalzell, Nikitas Stamatopoulos, Grant Salton, Mario Berta, and William J Zeng. Quantum resources required to block-encode a matrix of classical data. *IEEE Transactions on Quantum Engineering*, 3:1–23, 2022.

[16] Christopher M. Dawson and Michael A. Nielsen. The Solovay-Kitaev algorithm. *Quantum Info. Comput.*, 6(1):81–95, January 2006.

[17] Yulong Dong, Xiang Meng, K Birgitta Whaley, and Lin Lin. Efficient phase-factor evaluation in quantum signal processing. *Physical Review A*, 103(4):042419, 2021.

[18] Timothy N Georges, Bjorn Berntson, Christoph Sunderhauf, and Aleksei V Ivanov. Pauli decomposition via the fast Walsh-Hadamard transform. *New Journal of Physics*, 2024.

[19] Craig Gidney. How to write the iSWAP unitary as a linear combination of tensor products between 1-qubit gates? StackOverflow, 2023. `https://quantumcomputing.stackexchange.com/a/31790`.

[20] Brett Giles and Peter Selinger. Exact synthesis of multiqubit clifford+t circuits. *Phys. Rev. A*, 87:032332, Mar 2013.

[21] András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum Singular Value Transformation and beyond: exponential improvements for quantum matrix arithmetics. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 193–204, 2019.

[22] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Physical Review Letters*, 100(16):160501, 2008.

[23] Niels Gleinig and Torsten Hoefler. An efficient algorithm for sparse quantum state preparation. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 433–438. IEEE, 2021.

[24] Daniel Gunlycke, Mark C Palenik, Sean A Fischer, and Amie R Emmert. Efficient algorithm for generating Pauli coordinates for an arbitrary linear operator. *arXiv preprint arXiv:2011.08942*, 2020.

[25] Hiroki Hamaguchi, Kou Hamada, and Nobuyuki Yoshioka. Handbook for Quantifying Robustness of Magic. *Quantum*, 8:1461, September 2024.

[26] Lukas Hantzko, Lennart Binkowski, and Sabhyata Gupta. Tensorized Pauli decomposition algorithm. *Physica Scripta*, 99(8):085128, 2024.

[27] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15):150502, 2009.

[28] Raban Iten, Roger Colbeck, Ivan Kukuljan, Jonathan Home, and Matthias Christandl. Quantum circuits for isometries. *Physical Review A*, 93(3):032318, 2016.

[29] Tyson Jones. Decomposing dense matrices into dense Pauli tensors. *arXiv preprint arXiv:2401.16378*, 2024.

[30] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Fast and efficient exact synthesis of single qubit unitaries generated by Clifford and T gates. *arXiv preprint arXiv:1206.5236*, 2012.

[31] Océane Koska, Marc Baboulin, and Arnaud Gazda. A tree-approach Pauli decomposition algorithm with application to quantum computing. In *ISC High Performance 2024 Research Paper Proceedings (39th International Conference)*, pages 1–11. Prometeus GmbH, 2024.

[32] Robin Kothari. Efficient algorithms in quantum query complexity. 2014.

[33] Parker Kuklinski and Benjamin Rempfer. S-FABLE and LS-FABLE: Fast approximate block-encoding algorithms for unstructured sparse matrices. *arXiv preprint arXiv:2401.04234*, 2024.

[34] Luke Lapworth. A hybrid quantum-classical CFD methodology with benchmark HHL solutions. *arXiv preprint arXiv:2206.00419*, 2022.

[35] Zexian Li, Xiao-Ming Zhang, Chunlin Yang, and Guofeng Zhang. Binary tree block encoding of classical matrix. *arXiv preprint arXiv:2504.05624*, 2025.

[36] Lek-Heng Lim. Tensors in computations. *Acta Numerica*, 30:555–764, 5 2021.

[37] Lin Lin and Yu Tong. Optimal polynomial based quantum eigenstate filtering with application to solving quantum linear systems. *Quantum*, 4:361, 2020.

[38] Guang Hao Low and Isaac L Chuang. Hamiltonian simulation by qubitization. *Quantum*, 3:163, 2019.

[39] Guang Hao Low, Vadym Kliuchnikov, and Luke Schaeffer. Trading T gates for dirty qubits in state preparation and unitary synthesis. *Quantum*, 8:1375, 2024.

[40] Emanuel Malvetti, Raban Iten, and Roger Colbeck. Quantum circuits for sparse isometries. *Quantum*, 5:412, 2021.

[41] John M Martyn, Zane M Rossi, Andrew K Tan, and Isaac L Chuang. Grand unification of quantum algorithms. *PRX Quantum*, 2(4):040203, 2021.

[42] Liron Mor-Yosef, Shashanka Ubaru, Lior Horesh, and Haim Avron. Multivariate trace estimation using quantum state space linear algebra. *SIAM Journal on Matrix Analysis and Applications*, 46(1):172–209, 2025.

[43] Mikko Möttönen, Juha J Vartiainen, Ville Bergholm, and Martti M Salomaa. Quantum circuits for general multiqubit gates. *Physical Review Letters*, 93(13):130502, 2004.

[44] Mikko Möttönen, Juha J. Vartiainen, Ville Bergholm, and Martti M. Salomaa. Transformation of quantum states using uniformly controlled rotations. *Quantum Info. Comput.*, 5(6):467–473, September 2005.

[45] JA Oteo and J Ros. A fractal set from the binary reflected gray code. *Journal of Physics A: Mathematical and General*, 38(41):8935, 2005.

[46] Martin Plesch and Caslav Brukner. Quantum-state preparation with universal gate decompositions. *Physical Review A*, 83(3):032302, 2011.

[47] S. V. Romero and J. Santos-Suárez. PauliComposer: Compute tensor products of Pauli matrices efficiently. *Quantum Information Processing*, 22(12):449, 2023.

[48] Neil J. Ross and Peter Selinger. Optimal ancilla-free clifford+t approximation of z-rotations. *Quantum Information & Computation*, 16(11&12):901–953, 2016.

[49] Travis Russell. The exact synthesis of 1-and 2-qubit clifford+ t circuits. *arXiv preprint arXiv:1408.6202*, 2014.

[50] Peter Selinger. Generators and relations for n-qubit Clifford operators. *Logical Methods in Computer Science*, 11, 2015.

[51] Vivek V Shende, Stephen S Bullock, and Igor L Markov. Synthesis of quantum logic circuits. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, pages 272–275, 2005.

[52] Vivek V Shende, Igor L Markov, and Stephen S Bullock. Smaller two-qubit circuits for quantum communication and computation. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, volume 2, pages 980–985. IEEE, 2004.

[53] Yigit Subasi, Rolando D Somma, and Davide Orsucci. Quantum algorithms for systems of linear equations inspired by adiabatic quantum computing. *Physical Review Letters*, 122(6):060504, 2019.

[54] Xiaoming Sun, Guojing Tian, Shuai Yang, Pei Yuan, and Shengyu Zhang. Asymptotically optimal circuit depth for quantum state preparation and general unitary synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(10):3301–3314, 2023.

[55] Christoph Sünderhauf, Earl Campbell, and Joan Camps. Block-encoding structured matrices for data input in quantum computing. *Quantum*, 8:1226, 2024.

[56] Souichi Takahira, Asuka Ohashi, Tomohiro Sogabe, and Tsuyoshi Sasaki Usuda. Quantum algorithms based on the block-encoding framework for matrix functions by contour integrals. *arXiv preprint arXiv:2106.08076*, 2021.

[57] Juha J Vartiainen, Mikko Möttönen, and Martti M Salomaa. Efficient decomposition of quantum gates. *Physical Review Letters*, 92(17):177902, 2004.

[58] Jun Ying, Jun Shen, Li Zhou, Wei Zhong, Minghui Du, and Yu Sheng. Preparing a fast Pauli decomposition for variational quantum solving linear equations. *Annalen der Physik*, 535(11):2300212, 2023.

[59] Pei Yuan and Shengyu Zhang. Optimal (controlled) quantum state preparation and improved unitary synthesis by quantum circuits with any number of ancillary qubits. *Quantum*, 7:956, 2023.

[60] Xiao-Ming Zhang, Tongyang Li, and Xiao Yuan. Quantum state preparation with optimal circuit depth: Implementations and applications. *Physical Review Letters*, 129(23):230504, 2022.