Methods to simulate fermions on quantum computers with hardware limitations

Steudtner, M.

Methods to simulate fermions on quantum computers with hardware limitations

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus prof. mr. C.J.J.M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op woensdag 20 november 2019
klokke 15.00 uur

door

Mark Steudtner

geboren te Löbau, Duitsland in 1991

Promotores: prof. dr. S. D. C. Wehner (TU Delft)
prof. dr. C. W. J. Beenakker
Promotiecommissie: dr. J. D. Whitfield (Dartmouth College, VS)
dr. ir. M. Veldhorst (TU Delft)
prof. dr. K. J. Schoutens (UvA)
prof. dr. E. R. Eliel
prof. dr. K. E. Schalm

Casimir PhD Series Delft-Leiden 2019-33
ISBN 978-90-8593-416-5
An electronic version of this thesis can be found
at https://openaccess.leidenuniv.nl
This work was supported by the Netherlands Organization for Scientific Research (NWO/OCW) and an ERC Synergy Grant.

Contents

Chapter 1

Introduction

1.1 Preface

It is believed that quantum computers will help to increase our knowledge about large molecules and strongly correlated materials. There are many systems in those classes that we do not understand to this day, not for an inability to comprehend their physics, but because we lack computational power. In fact, we have accurate models describing the interactions of electrons in molecules, but classical computers are incompatible in dealing with the quantum properties. While we can easily formulate the Hamiltonians describing the systems, their spectrum and eigenstates (in particular, their ground state), that would allow us insight into their nature, can only be obtained with tremendous efforts. The issue is that with electrons being quantum-mechanical particles, their eigenstates are superpositions of all their possible configurations within the host system. As a consequence, any computer would need to have enough memory to encode the entirety of the electronic Hilbert space. Since the number of configurations is typically exponential in the size of the considered system, the same scaling applies to the resources required for classical simulation. This is a prohibitive overhead that not only places a cap on the number of active electrons we are able to simulate, but also on the simulation accuracy, since increasing the basis set is a means to make a simulation more precise. Fortunately, electronic systems can be stored differently on quantum computers, and so Abrams, Lloyd and Feynman suggested their use as universal quantum simulators [1–3].

A quantum computer is a device with a controllable quantum system as a

memory. As such, the memory spans an exponential Hilbert space which can imitate the exponential Hilbert space of an arbitrary electronic system. The required memory, which is the minimum number of qubits in the quantum computer, is thus only linear in the size of the problem. Not only are quantum states stored more adequately in qubits, but their manipulation by quantum gates, the elementary operations of the quantum computer, allows simulating arbitrary quantum-mechanical time evolutions. This approach, in which quantities of interest are calculated using gate operations rather than only the time evolution of the parent system, is commonly referred to as digital quantum simulation. While being technically more challenging than its analogue counterpart, it allows us to employ powerful quantum algorithms for the simulation and is compatible with techniques of quantum error mitigation and correction. However, since quantum devices have stricter limitations than classical computers, the feasibility of these amazing algorithms is determined by the right ‘programming’ of the quantum computer, which leads us to the subject matter of this thesis. Within these pages, we are going to deal with three limitations of quantum computing hardware in particular.
Firstly, there is the issue of memory size. Being controllable quantum systems, qubits are not easy to fabricate (or find) and must therefore be regarded as a rare commodity. The era of noisy intermediate scale quantum (NISQ) computers, for instance, is said to open with the arrival of devices containing only around fifty qubits [4]. While the amount of classical memory required to store the Hilbert space of those devices is enormous, the amount of qubits is not. Note also that many promising platforms are currently far from this intermediate scale.

Another issue is the qubit lifetime. Since quantum information can only be stored fleetingly before its decoherence, feasible algorithms have to be short. Quantum computation is thus strongly dependent on our ability to reduce the runtime of quantum algorithms by applying different parts of them in parallel. Not only has the need for parallelization an impact on algorithm design and compilation, but also on the device itself. Quantum algorithms are often expressed as quantum circuit diagrams, in which gate sequences can be grouped into parallel layers, but there is no guarantee that the parallel execution of such a layer is actually possible on the hardware level. Even in fault tolerant quantum computers, where the effect of decoherence is often argued to be diminished, it never really vanishes. These devices, that are further developed than NISQ computers, continuously run cycles of error correction algorithms that would allow for longer algorithms and so seemingly relax the need for a parallel operation. However, since the cycles are not exempt from noise, they also benefit from parallelization. In fact, when error correction cycles cannot be run within a certain time span, they cause errors rather than correct them.

Lastly, a quantum device has much stricter geometric constraints. This is due to the fact that the actual qubit part of the quantum computer has to be placed inside a dilution refrigerator shielding it from thermal fluctuations. Geometrical constraints not only limit the number of qubits, but also their connectivity and control.

Running a simulation algorithm, all of these peculiarities of quantum hardware need to be taken into account. Let us take a look on how these constraints can enter and influence a computation. Quantum computing is conceptually different from its classical counterpart, even though there are similarities in the way they are applied: for a classical computation, a user would pass problem-specific arguments to pre-existing, standard routines to obtain a result. When concerned with a ground state problem, for instance, a user would initially choose a matrix representation of the Hamiltonian in order to pass it to a diagonalization routine.

The situation is quite similar for digital quantum simulation, where algorithms like *quantum phase estimation* or the *variational quantum eigensolver* [5–7] can be regarded as predefined routines a user has to feed with problem-specific input. Since the problem is finding the ground state the input is, like in the classical case, a representation of the Hamiltonian. It can however not be a matrix, since its sheer dimension would not make it through classical preprocessing. Instead, the Hamiltonian is submitted in a form that the quantum computer can process, as a sum of operators acting on the exponential Hilbert space. To run quantum algorithms, the computer must be able to implement those operations as black box routines on its qubits and they therefore take the form of low-level quantum gates. Typically, the input Hamiltonian is a sum of weighted Pauli strings, where each of them is a product of Pauli operators on different qubits with a real coefficient. Note that with this construction the quantity preventing us from simulating arbitrarily-sized systems is no longer the matrix dimension, but the number of terms in the qubit Hamiltonian. The Pauli strings and their associated coefficients are subsequently turned into gate instructions in the quantum algorithm approximating the ground state. Algorithms based on *Trotterization* or *qubitization*, for instance, require that Pauli strings of the Hamiltonian are applied to the system directly [8–10], while in variational quantum eigensolvers they are measured on the memory. In this way, the input Hamiltonian will determine the performance of the applied algorithm, and the number of qubits required for the computation. In order to gain control over those quantities, an amount of pre-processing is required. A visualization of the entire quantum simulation process can be found in Figure 1.1, starting from a fermionic Hamiltonian in second quantization. Considerations about the basis set and truncation of the problem are inherent in this operator, which is to be regarded as a fixed quantity. However, as long as it is a valid representation of the fermionic Hamiltonian, the qubit Hamiltonian can be chosen freely. With that choice, we can influence the performance and requirements of the quantum algorithm that the qubit Hamiltonian feeds into. The transform between the fermionic operators and gate instructions, as well as the correspondence between fermionic occupations and qubit configurations, will be referred to as *fermion-to-qubit mapping*. Since a mapping directly relates the fermionic problem to the computer's memory, it would ideally be tailored to its device. The transform of the Hamiltonians, depicted in Figure 1.1, can be done using classical software readily available [11–13].

1.2 Fermion-to-qubit mappings

Definitions – Qubit basis and Pauli operators

§1 *The state of a single qubit shall be a linear combination of the basis states $|0\rangle$ and $|1\rangle$, which we will also refer to as computational basis. The (single-qubit) Pauli operators X, Y and Z shall be defined such that they are hermitian and unitary.*

$$Z = |0\rangle\langle 0| - |1\rangle\langle 1| \tag{1.1}$$

$$X = |1\rangle\langle 0| + |0\rangle\langle 1| \tag{1.2}$$

$$Y = i\,|1\rangle\langle 0| - i\,|0\rangle\langle 1| \tag{1.3}$$

§2 *To characterize the basis of n qubits, we are going to introduce binary vectors $\boldsymbol{\omega} = (\omega_1, \omega_2, \dots, \omega_n)^\top$, where each component ω_i is a binary number:*

$\omega_i \in \{0,1\} =: \mathbb{Z}_2$. We thus write $\boldsymbol{\omega} \in \mathbb{Z}_2^{\otimes n}$. For convenience, we are going to use a binary (modular) addition when adding two such binary vectors: $\boldsymbol{\omega} + \boldsymbol{\mu} = \bigotimes_{i=1}^{n}(\omega_i + \mu_i \bmod 2)$. The modulus shall be implicitly applied also to matrix and scalar products of binary vectors. Within this thesis we use those vectors to define the multi-qubit computational basis states as

$$|\boldsymbol{\omega}\rangle \;=\; \bigotimes_{i=1}^{n} |\omega_i\rangle \;=\; |\omega_1\rangle \otimes |\omega_2\rangle \otimes \cdots \otimes |\omega_n\rangle \;, \tag{1.4}$$

such that an n-qubit quantum state $|\varphi\rangle$ takes the generic form

$$|\varphi\rangle \;=\; \sum_{\boldsymbol{\omega}\in\mathbb{Z}_2^{\otimes n}} a_{\boldsymbol{\omega}} \, |\boldsymbol{\omega}\rangle \;, \tag{1.5}$$

where there are 2^n complex, normalized coefficients $a_{\boldsymbol{\omega}}$: $\sum_{\boldsymbol{\omega}} |a_{\boldsymbol{\omega}}|^2 = 1$.

§3 *Products of Pauli operators will be referred to as Pauli strings. To distinguish operators* (1.1)-(1.3) *acting on different qubits, we brand them with qubit indices. The operator X_i for instance acts as an X-operator on qubit i, and as the identity ($\mathbb{I}$) on the rest. In general, we want to omit single-qubit identities and use the following shorthand for Pauli strings $Z \otimes \mathbb{I} \otimes Z = Z_1 \otimes Z_3 = \bigotimes_{i\in\{1,3\}} Z_i$, though we will use $\mathbb{I}$ for the identity operation on the entire system. A qubit Hamiltonian is typically a weighted sum of Pauli strings. In the second chapter, for instance, we find the following Hamiltonian for the hydrogen molecule in a minimal basis (for atoms at their bond distance, in units of Hartree):*

$$\begin{aligned} H \;=\; & -0.34\,\mathbb{I} \;+\; 0.18129\,X_1 \otimes X_2 \;+\; 0.394\,Z_1 \\ & +\; 0.0112\,Z_1 \otimes Z_2 \;+\; 0.394\,Z_2 \,. \end{aligned} \tag{1.6}$$

Definitions – Fermionic operators

§4 *Within this thesis, we denote fermionic creation and annihilation operators by $c_j^\dagger$ and c_j, where j denotes the index of the fermion orbital, the combination of spatial wave function and spin indices. When speaking outside a context of electrons, we also refer to orbitals as fermionic modes, in particular when we replace those indices with coordinates on a hypothetical, two-dimensional embedding of the system. To account for their fermionic* *nature, creation and annihilation operators satisfy the following anticommutation relations:*

$$\left[c_i, c_j\right]_+ = 0, \quad \left[c_i^\dagger, c_j^\dagger\right]_+ = 0, \quad \left[c_i, c_j^\dagger\right]_+ = \delta_{ij} \; , \tag{1.7}$$

where $[A, B]_+ = AB + BA$.

§5 *To avoid confusion with qubit configurations, we will denote the vacuum state, in which all fermion modes (orbitals) are unoccupied, by* $|\Theta\rangle$. *States are subsequently defined as products like* $c_{i_1}^\dagger c_{i_2}^\dagger ... c_{i_M}^\dagger \, |\Theta\rangle$.

§6 *While second quantization makes for some redundancy in the ordering of the creation operators, it allows us to express many-body Hamiltonians in a concise form. The Hamiltonians of the electronic structure problem, that we are interested in, are following the pattern of*

$$\underbrace{\sum_{ij} t_{ij} \; c_i^\dagger c_j}_{\substack{\text{kinetic / hopping} \\ \text{and nuclear terms}}} \quad + \quad \underbrace{\sum_{ijkl} V_{ijkl} \; c_i^\dagger c_j^\dagger c_k c_l}_{\text{two-body interactions}} \; , \tag{1.8}$$

where the sums extend over all mode indices. The coefficients t_{ij} *and* V_{ijkl} *are numbers (integrals depending on the chosen basis) that are generally complex, but related with respect to their indices such that the Hamiltonian is hermitian. Note that by the structure of* (1.8), *all of its terms conserve the number of particles of a state* $c_{i_1}^\dagger c_{i_2}^\dagger \; ... \; c_{i_M}^\dagger \, |\Theta\rangle$, *and most of the terms commute with one another.*

Let us take the qubit requirements as an example of mappings dealing with hardware limitations. The number of qubits storing the problem's Hilbert space is determined by the Hamiltonian: to solve a problem, we need as many qubits as the Hamiltonian acts on. For devices with few (logical) qubits, we are interested in keeping these qubit requirements to a minimum which is determined by the problem's degrees of freedom. Neglecting symmetries like the conservation of particles, mappings typically use more qubits than absolutely necessary. Indeed we need minimally 16 qubits to encode all possible occupations of 16 fermionic modes,

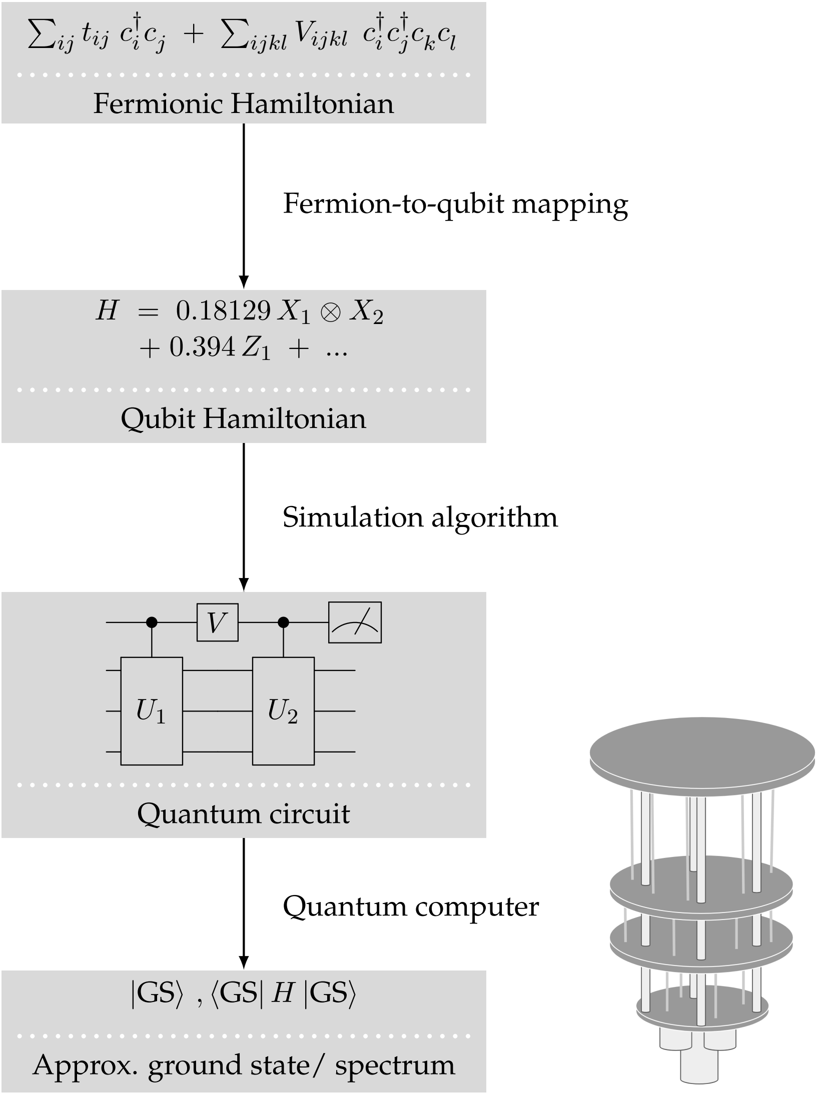

$\sum_{ij} t_{ij}\, c_i^\dagger c_j \;+\; \sum_{ijkl} V_{ijkl}\; c_i^\dagger c_j^\dagger c_k c_l$
Fermionic Hamiltonian
Fermion-to-qubit mapping
$H \;=\; 0.18129\, X_1 \otimes X_2$
$+\, 0.394\, Z_1 \;+\; ...$
Qubit Hamiltonian
Simulation algorithm
V
U_1
U_2
Quantum circuit
Quantum computer
$|\mathrm{GS}\rangle\;, \langle \mathrm{GS}|\, H\, |\mathrm{GS}\rangle$
Approx. ground state/ spectrum

Figure 1.1. Simulating fermions on a quantum computer, depicted as a flow chart. The process starts with a problem Hamiltonian and results in the ground state and its energy being approximated by a quantum computer. However, that ground state problem refers not to the initial Hamiltonian, but rather to a prudently chosen qubit version of it. In combination with the simulation algorithm, the qubit Hamiltonian then determines the quantum circuit, that can be compiled into a program that the quantum computer runs.

but typically not all of those configurations are relevant. Information about the number of fermions in the system is usually given besides the Hamiltonian as either a constraint or from real-life observation of the system. Let us say there is only one particle hopping around in the system given, then the basis of the problem is spanned by 16 configurations with the fermion sitting in a different mode in every basis state. Following combinatorial arguments, these degrees of freedom could be encoded by 4 qubits, and so 12 qubits saved. This of course just an example, but a relevant one since for all particle-number conserving problems, modes are at most half filled (with particles or holes). While encoding all particle numbers usually yields a, in some sense, simpler Hamiltonian, one might be interested in bringing the required number of qubits closer to the minimum. This is the subject of the second chapter of this thesis, in which we consider the impact of classical code layers on fermion-to-qubit mappings. Typically, improvements in the qubit number will have some negative influence on the Hamiltonian. As we will see in the second chapter, this manifests in the choice between either making the gates in the Hamiltonian terms more complex, or accepting a larger number of Pauli string terms instead. Clearly, trading qubits is expensive in the runtime of the simulation algorithm. In fact, one might even consider employing more qubits if only the algorithmic performance could be improved. At least to some degree, this turns out to be possible. While it seems difficult to reduce the number of terms in a Hamiltonian, we can at least make sure they act on the qubit system in a way that would allow us to parallelize the algorithm.

This path is taken in the third chapter, where fermion-to-qubit mappings are enhanced by adding a quantum code layer, for which a number of additional qubits is required. While in the second chapter we have been considering small quantum devices, the focus is now shifted to devices with a larger number of qubits. However we do not want to forgo the aforementioned limitations and assume those devices to contain an unlimited amount of qubits. Instead, let us think of them as being from the NISQ era. With that in mind we will cling to two reasonable strategies. Firstly, we have to compromise between resource requirements and parallelization. With additional qubits to spare, one might for instance be tempted to encode the interactions of the fermionic Hamiltonian directly, but the problem with that is their number, and therefore the number of qubits, usually grows much faster with the system size. Secondly, due to the connectivity, the quantum device is not going to be able to perform operations between any two qubits. Thus simulated Hamiltonians should ideally only have terms involving coupled qubits, that can be entangled without disturbing others. The quantum codes used in the fermion-to-qubit mappings should thus be defined locally on a realistic layout for a quantum device.

The quantum codes found in the third chapter incorporate both of these strategies: they are local and planar on a square lattice and require a number of qubits that scales with the system size, rather than the Hamiltonian. Note that the square lattice appears to be a natural choice, as it is also the canvas for surface code spurring efforts to build transmon chips in this layout. While we focus on mappings that provide a geometrical embedding of qubit Hamiltonians, their codes would also allow for the application of error mitigation techniques. While those strategies might help improve results in the short term, scalable simulation algorithms are believed to require quantum error correction. The difference between the two is that error mitigation is aiming to filter noise from the obtained data, whereas with error correction, one is aiming to prevent errors during the computation. Practically, quantum error correction codes would constitute a code layer that is unlike the codes in the third chapter. In quantum error correction, physical gate instructions are replaced with their logical counterparts and quantum circuits with their fault-tolerant versions. The entire computation embedded in error correction cycles, in which stabilizers are measured and syndromes are extracted. The problem is that running such cycles is technically challenging even without a computation happening in between.

1.3 Quantum error correction

Definitions – Quantum error correction codes

§7 *A quantum stabilizer code is a mapping between two systems with a different qubit number, where the smaller system is encoded by the larger one. Let us say that the smaller system has n_1 qubits, and the larger one n_2, then a $[[n_2, n_1, d]]$ quantum code maps every state of the former system $|\varphi\rangle$ to a state on the larger system $|\overline{\varphi}\rangle$: $|\varphi\rangle \mapsto |\overline{\varphi}\rangle$. The integer d is called code distance, and will be explained in §10. For the encoding to be a one-*

to-one mapping, these quantum codes constrain $n_2 - n_1$ qubits worth of degrees of freedom by so-called stabilizer conditions. That is, there is a set of commuting Pauli strings such that for each member S we find

$$S\,|\overline{\varphi}\rangle \;=\; |\overline{\varphi}\rangle\;, \tag{1.9}$$

for all encoded states $|\overline{\varphi}\rangle$. S is called a stabilizer. Considering that the identity is part of the stabilizer set, it forms a group generated by $n_2 - n_1$ Pauli strings (using the operator product).

§8 *Not just the states, but also the operators of the smaller system are encoded. For every physical operator $\mathcal{O}$ in the smaller, there exist 'logical' operators $\overline{\mathcal{O}}$ in the larger system, such that*

$$\mathcal{O}\,|\varphi\rangle \;\mapsto\; \overline{\mathcal{O}}\,|\overline{\varphi}\rangle\;. \tag{1.10}$$

Note that there are several logical operators for one physical operator, since their action on the code space is equivalent by the multiplication with stabilizers, i.e. $\overline{\mathcal{O}}$ and $S \cdot \overline{\mathcal{O}}$ have the same effect. Logical operators generally preserve the encoded subspace, i.e. they commute with the stabilizers.

§9 *Stabilizer codes are the workhorse of quantum error correction, since Pauli errors that might occur in a noisy device can either be identified when they anticommute with stabilizers, or are stabilizers themselves and their action therefore trivial. Continuously measuring stabilizers, the system is projected into the subspaces corresponding to outcomes $\langle S\rangle = \pm 1$. Flipped expectation values, referred to as syndromes, can then be attempted to be corrected.*

§10 *With $[[n_2,\, n_1,\, d]]$ error correction codes, one can correct for all conceivable Pauli errors up to a certain weight (the number of nontrivial Pauli operators in the string). Assuming that lower-weight Pauli errors occur with a much higher rate, quantum information is preserved by being stored nonlocally. It is thus unsurprising that the maximal weight of correctable Pauli errors is connected to the minimal weight of logical operators, which is the code distance d. While we can always correct for errors of weight up to $(d-1)/2$, they can be detected up to a weight of $d-1$. In the latter case, the errors that occurred can no longer be discerned, but syndromes may still serve as an indication to discard the outcome of this particular computation. In an error correction setting, Pauli errors with a weight higher than $(d-1)/2$ will generally cause an error on the logical system,* *such that the physical error rate is translated into a logical one. An error correction code is of course only useful if the latter is lower then the former, but there is a threshold of physical errors above which the code causes the rate to increase. Is the noise of a device above this threshold, the goal of fault tolerance, to decrease the logical noise until it becomes negligible, cannot be achieved.*

Definitions – Surface code

§11 *An important example of quantum error correction codes is the surface code. The code is popular due to its high threshold [14] and the availability of efficient decoding algorithms [15]. Surface code is the planar version of Kitaev's toric code [16], that in its 'rotated' version [17], is an* $[[d^2, 1, d]]$ *code for an arbitrary odd-valued distance. The* $d = 3$ *and* $d = 5$ *version of the code, as well as the logical operators are depicted in Figure 1.2(b)-(d). Excluded from that count are* $d^2 - 1$ *measurement qubits, that can help perform the syndrome measurements fault-tolerantly, see Figure* 1.2(a). *The code is planar on a square lattice, where the stabilizer generators are overlapping plaquettes that have the structure of* $Z^{\otimes 4}$ *and* $X^{\otimes 4}$. *With the logical operators* $\overline{X}$ *and* $\overline{Z}$ *being defined as* Z*- and* X*-strings from one boundary to the other, the protection of the logical qubit increases with the diameter of the code patch.*

Quantum error correction not only requires many physical qubits, but also precise operations on all of them in parallel. At this point, engineering problems clash with theoretical proposals. In the third chapter we consider two operations parallelizable if they do not use common resources like qubits and couplers (or whatever mediates two-qubit gates), but for real quantum devices, even that is not entirely true. In reality, processes (that for instance implement quantum gates) might not just share quantum resources, but also their control elements: let us say that the qubits are connected to their classical control by some sort of lines. While there has to be a method to select one individual qubit for a quantum gate, it is naïve to assume that a line would not be connected to a number

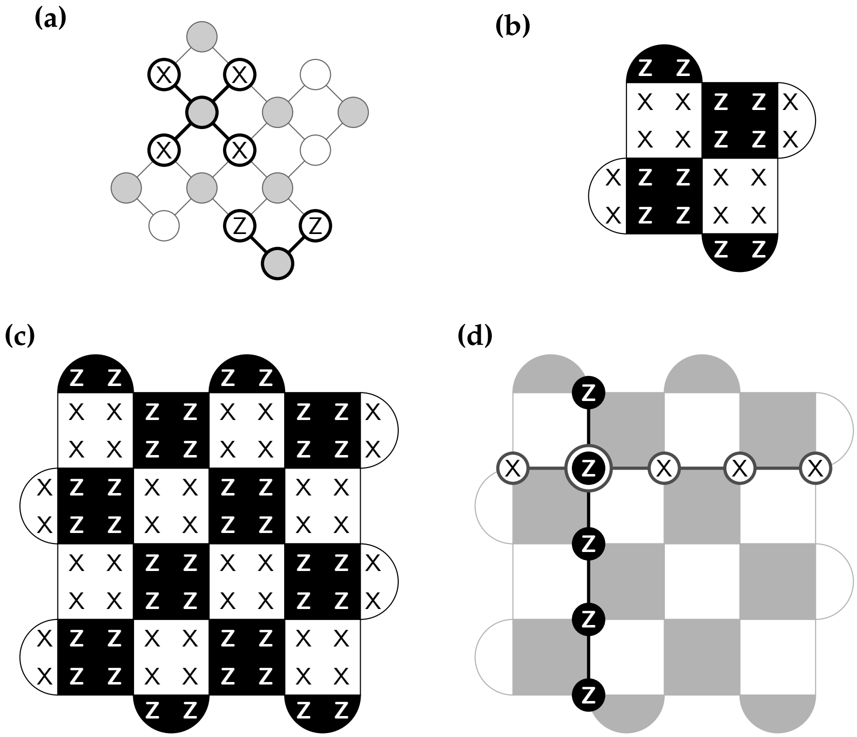


Figure 1.2. Rotated surface code. **(a)** Connectivity graph of the distance-three code. Two stabilizers are highlighted, where for each stabilizer the parity of the involved physical qubits (white), in Z- or X-basis, is collected on the measurement qubit (gray). For that purpose, two-qubit gates have to be performed along the highlighted edges. **(b) & (c)** Stabilizer tiles for the distance three and five code, where every tile is a separate stabilizer with the Pauli operators close to the location of the physical qubits. **(d)** Logical operators of the distance-five code. For rotated surface code, logical operators are Pauli strings across the code patch. $\overline{X}$ and $\overline{Z}$ operators in the figure overlap on exactly one qubit, on which one acts as X, the other as Z such that the logicals anticommute with each other, but commute with all stabilizers.

of qubits at once. Given the sheer amount of qubits, not all of them can possibly be wired individually. In a more realistic scheme, each qubit would have several line contacts and an individual interaction would take place only when multiple of them are operated, see Figure 1.3(a). This allows for individual qubit control, but ultimately has repercussions on parallelization. Imagine two operations in different places where the control lines used for the first and second operation happen to interface at another qubit that is to be left unaffected. Unfortunately, this is only guaranteed as long as the two operations are done sequentially, as parallelizing them will have spurious effects on the qubit at the crossing, shown in Figure 1.3(b). This example is not only far from being unlikely but generalizes into a major drawback of the scheme. However, the reduction of classical control architecture on the quantum device could be regarded as more important. A scalable architecture is a prerequisite for bringing as many qubits as possible onto the same device and into a fridge. As long as those refrigerators do not grow substantially in size over the next years, it will not alone be the amount of control elements to determine whether fault tolerance can be achieved, but also the spatial size of the qubits. The qubit density is of course specific to each platform, and while transmons are a popular technology at the moment, they might eventually be made obsolete by spin qubits in semiconductor quantum dots, for which a much higher density is expected to be achieved [18].

To make use of this density, a crossbar architecture for shared control of spin qubits in silicon quantum dots is proposed in [18]. As we show in the fourth chapter, its limited control mechanisms are sufficient to run quantum error correction cycles without spurious effects. For that purpose we introduce a model of how the quantum dot processor can be controlled from the periphery of the chip. With the focus on bridging the gap between device operations (like control pulses) and quantum circuits, irrelevant details about the device physics are omitted in this model. Taking into account the peculiarities of the system, we discuss parallelization of the available gate operations while providing programmatic steps for the native implementation of surface and color codes. To achieve high fidelity gates, we even restrict the parallelization further. Still, our estimates suggest that a large enough system will be able to suppress the error such that it does not fail more often than a classical memory.

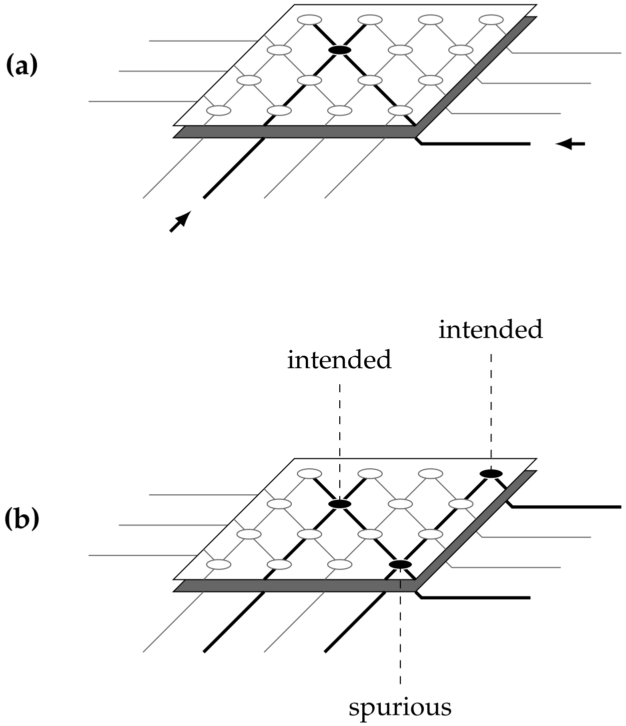


Figure 1.3. Shared control of elements on a chip. **(a)** Each control line connects to several elements. From the periphery, two lines (thickened) are operated to address a single element at their crossing. **(b)** When attempting to address two elements in parallel, the operated lines cross and can cause spurious effects at an unintended crossing.

1.4 This thesis

In this section, we will give a description of the objective in each chapter, and summarize our methods and findings. Each chapter's main contribution is specified. Before we start however, a few words on the structure of this thesis: the individual chapters are opened with a *Background* section and a more detailed discussion about its results. Each chapter's main text is followed by a section called *Supplement* holding additional information referenced in the text, and a table of relevant notations. Scientific progress does not stop while students write their theses, and so a *Further work* section is added to the second and third chapter, describing the latest developments on their respective subjects. This thesis ends with a *Summary* in english and in dutch, the *Curriculum Vitae* of its author and a list of publications.

1.4.1 Chapter two

In the second chapter, we will get in touch with the *Jordan-Wigner transform* [19] in its function as a fermion-to-qubit mapping. It is a simple method, in which each mode is assigned a qubit, indicating fermionic occupation when in the configuration $|1\rangle$. Let us consider the example of Figure 1.4(a), where the modes are represented by small bins and the balls that fill them represent fermions. Beneath the modes, a register of six qubits is shown encoding the state $c_1^\dagger c_2^\dagger c_5^\dagger \,|\Theta\rangle$. With the Jordan-Wigner transform, the entire Fock space of six orbitals can be encoded in states $|\boldsymbol{\nu}\rangle$ where $\boldsymbol{\nu}$ is any binary vector with, in this case, six components. However, a mere subset of vectors $\boldsymbol{\nu}$ is usually needed to describe the system, which means the Jordan-Wigner transform is inviting unphysical states. In fact, with only the physical states, a number of qubits could be saved. With the resource requirements in mind, we develop a framework for classical encodings of the Jordan-Wigner transform: the idea is to represent only the set of physical configurations $\boldsymbol{\nu}$, but use all possible configurations $|\boldsymbol{\omega}\rangle$ to do so. Encoding fewer degrees of freedom, such a mapping requires fewer qubits. Mathematically, it is defined by an encoding function $\boldsymbol{e}(\boldsymbol{\nu}) = \boldsymbol{\omega}$ and its inverse, the decoding. Since, in particular, the decoding function can be nonlinear with respect to its input $\boldsymbol{\omega}$, the operator transform is re-expressed in terms of a superoperator $\mathfrak{X}$, that can output gate instructions even for those codes. On code examples, we discuss the trade-offs between resource requirements and Hamiltonian size.

The main contribution of this chapter is a general approach for fermion-to-qubit mapping using classical codes.

1.4.2 Chapter three

In the third chapter, we touch upon the subject of the Jordan-Wigner transform of operators. We remark that physical Hamiltonians like (1.8) are build from products $c_i^\dagger c_j$, through which they conserve the number of particles. What terms like $c_i^\dagger c_j$ do is to move fermions from mode j to mode i, so the way to imitate this behavior for the Jordan-Wigner transform is to flip the corresponding qubits. However, since the $c_i^\dagger c_j$ may anticommute with creation operators of $c_{i_1}^\dagger c_{i_2}^\dagger ... c_{i_M}^\dagger \left|\Theta\right\rangle$, and so minus signs have to be accounted for. It follows that Pauli strings, translated from hopping terms like $c_i^\dagger c_j$, not only flip qubits i and j, but also scan for anticommutations using Z operators on all qubits in between them. In Figure 1.4(b) it is shown how a fermion hopping from modes 1 to 6 acquires two minus signs from passing particles in modes 2 and 5.
The Pauli string encoding this event is $X_1 \otimes Z_2 \otimes Z_3 \otimes Z_4 \otimes Z_5 \otimes X_6$. Note that this substring of Z-operators does not occur in local hoppings. However, this is a statement about locality in one dimension. In the third chapter, we are interested in harnessing the connectivity of two-dimensional qubit arrays for quantum simulation – a domain in which the Jordan-Wigner transform fails to map local interactions into local gates. A nonlocal string that results from a hopping interaction crossing several rows (when the modes are ordered along a winding pattern from row to row) is shown in Figure 1.4(c). As a one-dimensional fermion-to-qubit mapping, the Jordan-Wigner transform is bound to have nonlocal connections in two dimensions. However, two-dimensionality can be achieved by concatenation with a suitable quantum code. By adding auxiliary qubits and re-constraining them in stabilizers S, an arbitrary state of the memory (1.5) is mapped into something proportional to
$\left[\prod_S (1+S)\right] \left|\varphi\right\rangle \otimes \left|\chi\right\rangle$. Since the stabilizers S have an effect on $\left|\chi\right\rangle$, the configuration of the auxiliary qubits, one could argue that they store the effect S on $\left|\varphi\right\rangle$. By the non-uniqueness of logical operators, we multiply nonlocal strings with stabilizers S, which, if the stabilizers are chosen appropriately, leads to the cancellation of the entire Z-substring. What remains of the logical operator is a local term on the original qubits and a contribution on the auxiliaries. Ensuring these remainders are local is one

of the many tasks covered in the third chapter, in which we also generalize the notion of locality to long-range interactions. The main contribution of this chapter is a novel fermion-to-qubit mapping using quantum codes to manipulate operators.

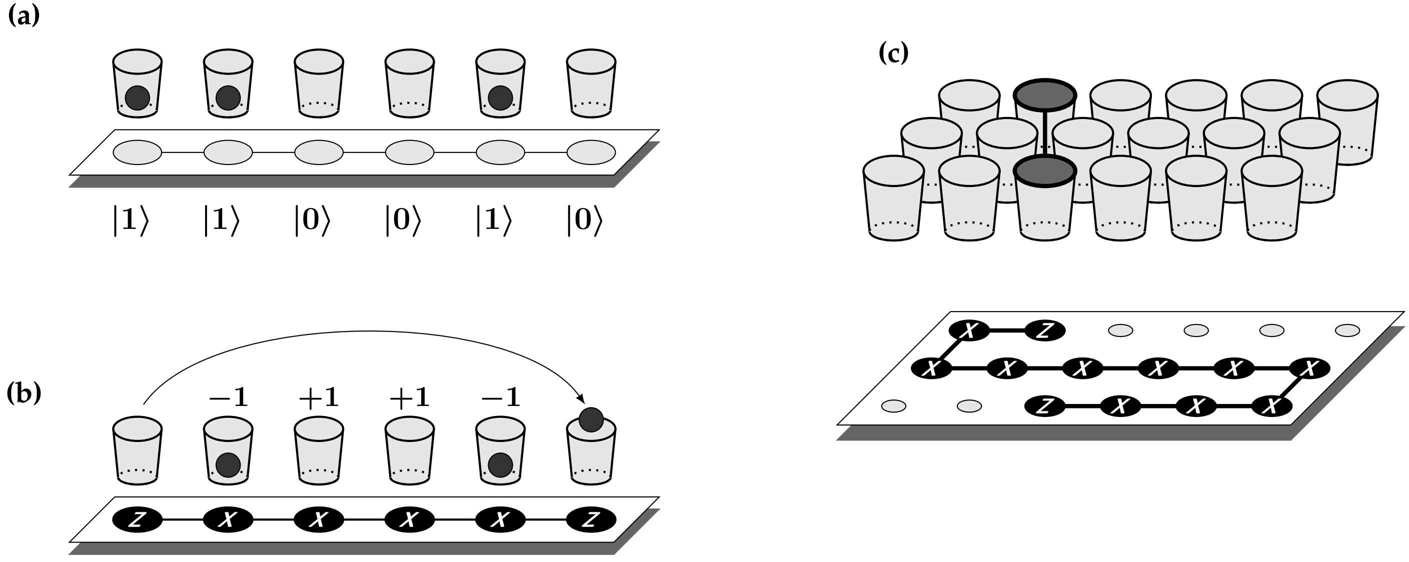


Figure 1.4. Jordan-Wigner transform. The bins represent fermionic modes, the balls fermions. The mapping to qubits on a chip is shown. **(a)** Basis transform with respect to a certain fermionic occupation. **(b)** Operator mapping. An exchange term, in which a fermion changes mode is mapped to a Pauli string in which the corresponding qubits are flipped. The fermion's anticommutations are denoted by '-1' signs and accounted for by Z-operators in the string. **(c)** Nonlocal Pauli strings resulting from a vertical hopping term in an S-pattern version of the Jordan-Wigner transform in two dimensions.

1.4.3 Chapter four

In the fourth chapter, we adapt surface and color code algorithms to the proposed architecture of [18], consisting of a grid of quantum dots on a silicon substrate and physical gates as sketched in Figure 4.1. A quantum dot is a physical site that can confine a single electron serving as a qubit. A square lattice of them is steered by shared control elements contacted from the grid's perimeter. Adjacent dots, for instance, are isolated by barrier gates extending horizontally and vertically over the grid. In this way, one barrier is shared by an entire row or column of dot pairs. To perform a two-qubit quantum gate, the barrier voltage on the barrier between the corresponding dot pairs must be changed, but the operation also affects all parallel dot pairs. This is however not all, since for two-qubit gates, the potential of one of the dots must be changed. The dot potential is manipulated via a type of physical gate, connecting quantum dots diagonally with respect to the direction of the barriers. Only where the diagonal crosses with the barrier, the two-qubit gate is performed. Also, the type and fidelity of the gates varies with the direction the dot pair is aligned. Fortunately, qubit positions can be changed by coherent shuttling [20], such that we can solely rely on high-fidelity gates. The shuttling is instigated by the manipulation of barrier and potentials, such that it can be performed by using the barrier and diagonal gate lines. Shuttling also has a part in performing single-qubit quantum gates at an individual address, since the system only allows to perform these operations globally, on exactly half of the dots. These operations are sufficient to run quantum error correction codes in a highly parallelized fashion, but a higher fidelity is expected when the operation are performed such that parallel two-qubit gates can have individual durations. All of these considerations are considered in the logical error analysis. The main contribution of this chapter is a road map for scalable quantum error correction in silicon quantum dots.

Chapter 2

Saving qubits with classical codes

2.1 Background

One essential component in realizing simulations of fermionic models on quantum computers is the representation of such models in terms of qubits and quantum gates. Following initial simulation schemes for fermions hopping on a lattice [3], more recent proposals used the Jordan-Wigner [19] transform [21–24], the Verstraete-Cirac mapping [25], or the Bravyi-Kitaev transform [26] to find a suitable representation. Specifically, the task of all such representations is two-fold. First, we seek a mapping from states in the fermionic Fock space of N sites to the space of n qubits. The fermionic Fock space is spanned by 2^N basis vectors $|\nu_1, \dots, \nu_N\rangle$ where $\nu_j \in \{0, 1\}$ indicates the presence ($\nu_j = 1$) or absence ($\nu_j = 0$) of a spinless fermionic particle at orbital j. Such a mapping $\mathrm{e} : \mathbb{Z}_2^{\otimes N} \mapsto \mathbb{Z}_2^{\otimes n}$ is also called an *encoding* [27]. An example of such an encoding is the trivial one in which $n = N$ and qubits are used to represent the binary string $\boldsymbol{\nu} = (\nu_1, \dots, \nu_N)^\top$. That is,

$$|\boldsymbol{\omega}\rangle = |\mathrm{e}\left(\boldsymbol{\nu}\right)\rangle = \bigotimes_{j=1}^{n} |\omega_j\rangle \ , \tag{2.1}$$

where $\omega_j = \nu_j$ in the standard basis $\{|0\rangle, |1\rangle\}$.

Second, we need a way to simulate the dynamics of fermions on these N orbitals. These dynamics can be modeled entirely in terms of the annihilation and creation operators c_j and $c_j^\dagger$ that satisfy the anti-commutation relations (1.7). Following these relations, the operators act on the fermionic Fock space as

$$c^\dagger_{i_m}\, c^\dagger_{i_1} \dots\, c^\dagger_{i_{m-1}} c^\dagger_{i_m} c^\dagger_{i_{m+1}} \dots\, c^\dagger_{i_M}\, |\Theta\rangle = 0 \tag{2.2}$$

$$c_{i_m}\, c^\dagger_{i_1} \dots\, c^\dagger_{i_{m-1}} c^\dagger_{i_{m+1}} \dots\, c^\dagger_{i_M}\, |\Theta\rangle = 0 \tag{2.3}$$

$$\begin{aligned} & c_{i_m}\, c^\dagger_{i_1} \dots\, c^\dagger_{i_{m-1}} c^\dagger_{i_m} c^\dagger_{i_{m+1}} \dots\, c^\dagger_{i_M}\, |\Theta\rangle \\ = & \,(-1)^{m-1}\, c^\dagger_{i_1} \dots\, c^\dagger_{i_{m-1}} c^\dagger_{i_{m+1}} \dots\, c^\dagger_{i_M}\, |\Theta\rangle \end{aligned} \tag{2.4}$$

$$\begin{aligned} & c^\dagger_{i_m}\, c^\dagger_{i_1} \dots\, c^\dagger_{i_{m-1}} c^\dagger_{i_{m+1}} \dots\, c^\dagger_{i_M}\, |\Theta\rangle \\ = & \,(-1)^{m-1}\, c^\dagger_{i_1} \dots\, c^\dagger_{i_{m-1}} c^\dagger_{i_m} c^\dagger_{i_{m+1}} \dots\, c^\dagger_{i_M}\, |\Theta\rangle\,, \end{aligned} \tag{2.5}$$

where $|\Theta\rangle$ is the fermionic vacuum and $\{i_1, \dots, i_M\} \subseteq \{1, \dots, N\}$. Mappings of the operators c_j to qubits typically use the Pauli matrices X, Z, and Y acting on one qubit, characterized by their anti-commutation relations $[P_i, P_j]_+ = 2\,\delta_{ij}\,\mathbb{I}$, for all $P_i \in \mathcal{P} = \{X, Y, Z\}$. An example of such a mapping is the Jordan-Wigner transform [19] given by

$$c_j \mathrel{\hat{=}} Z^{\otimes j-1} \otimes \sigma^- \otimes \mathbb{I}^{\otimes n-j} \tag{2.6}$$

$$c_j^\dagger \mathrel{\hat{=}} Z^{\otimes j-1} \otimes \sigma^+ \otimes \mathbb{I}^{\otimes n-j} \tag{2.7}$$

where

$$\sigma^- \;=\; |0\rangle\langle 1| = \frac{1}{2}\,(X + iY)\;, \tag{2.8}$$

$$\sigma^+ \;=\; |1\rangle\langle 0| = \frac{1}{2}\,(X - iY)\;. \tag{2.9}$$

It is easily verified that together with the trivial encoding (2.1) this transformation satisfies the desired properties (2.2)-(2.5) and can hence be used to represent fermionic models with qubit systems.

In order to assess the suitability of an encoding scheme for the simulation of fermionic models on a quantum computer, a number of parameters are of interest. The first is the total number of qubits n needed in the simulation. Second, we may care about the gate size of the operators c_j and $c_j^\dagger$ when mapped to qubits. In its simplest form, this problem concerns the total number of qubits on which these operators do not act trivially, that is, the number of qubits L, on which an operator acts as $P_j \in \mathcal{P}$ instead of the identity $\mathbb{I}$, sometimes called the Pauli length. Different transformations can lead to dramatically different performance with respect to these parameters. For both the Jordan-Wigner as well as the Bravyi-Kitaev transform $n = N$, but we have $L = O(n)$ for the first, while $L = O(\log n)$ for the second. We remark that in experimental implementations we typically do not only care about the absolute number L, but rather the specific gate size and individual difficulty of the qubit gates each of which may be easier or harder to realize in a specific experimental architecture. For error-corrected quantum simulation, the cost in T-gates is as important to optimize as the circuit depth [28], and quantum devices with restricted connectivity even require mappings tailored to them [29, 30]. Finally, we remark that instead of looking for a mapping for individual operators $c_j^{(\dagger)}$ we may instead opt to map pairs (or higher order terms) of such operators at once, or even look to represent sums of such operators.

2.2 Results

Here, we propose a general family of mappings of fermionic models to qubit systems and quantum gates that allow us to trade off the necessary number of qubits n against the difficulty of implementation as parametrized by L, or more complicated quantum gates such as CPHASE. Ideally, one would of course like both the number of qubits, as well as the gate size to be small. We show that our mappings can lead to significant savings in qubits for a variety of examples (see Table 2.1) as compared to the Jordan-Wigner transform for instance, at the expense of greater complexity in realizing the required gates. The latter may lead to an increased time required for the simulation depending on which gates are easy to realize in a particular quantum computing architecture.

At the heart of our efforts is an entirely general construction of the creation and annihilation operators in (2.2) given an arbitrary encoding $\mathbf{e}$ and the corresponding decoding $\mathbf{d}$. As one might expect, this construction is not efficient for every choice of encoding $\mathbf{e}$ or decoding $\mathbf{d}$. However, for linear encodings $\mathbf{e}$, but possibly nonlinear decodings $\mathbf{d}$, they can take on a very nice form. While in principle any classical code with the same properties can be shown to yield such mappings, we provide an appealing example of how a classical code of fixed Hamming weight [31] can be used to give an interesting mapping.

Two other approaches allow us to be more modest with the algorithmic depth in either accepting a qubit saving that is linear with N, or just saving a fixed amount of qubits for hardly any cost at all.
In previous works, trading quantum resources has been addressed for general algorithms [32], and quantum simulations [33–35]. In the two works of Moll et al. and Bravyi et al., qubit requirements are reduced with a scheme that is different from ours. A qubit Hamiltonian is first obtained with e.g. the Jordan-Wigner transform, then unitary operations are applied to it in order taper qubits off successively. The paper by Moll et al. provides a straightforward method to calculate the Hamiltonian, that can be used to reduce the amount of qubits to a minimum, but the number of Hamiltonian terms scales exponentially with the particle number. The notion that our work is based on, was first introduced in [34] by Bravyi et al., for linear en- and decodings. With the generalization of this method, we hope to make the goal of qubit reduction more attainable in reducing the effort to do so. The reduction method is mediated by nonlinear codes, of which we provide different types to choose from. The transform of the Hamiltonian is straight-forward from there on, and we give explicit recipes for arbitrary codes. We can summarize our contributions as follows.

- We show that for any encoding $\mathbf{e} : \mathbb{Z}_2^{\otimes N} \mapsto \mathbb{Z}_2^{\otimes n}$ there exists a mapping of fermionic models to quantum gates. For the special case that this encoding is linear, our procedure can be understood as a slightly modified version of the perspective taken in [27]. This gives a systematic way to employ classical codes for obtaining such mappings.

- Using particle-conservation symmetry, we develop 3 types of codes that save a constant, linear and exponential amount of qubits (see Table 2.1 and Sections 2.4.3.1-2.4.3.3). An example from classical coding theory [31] is used to obtain significant qubit savings (here called the binary addressing code), at the expense of increased gate difficulty (unless the architecture would easily support multi-controlled gates).

- The codes developed are demonstrated on two examples from quantum chemistry and physics.

- The Hamiltonian of the well-studied hydrogen molecule in minimal basis is re-shaped into a two-qubit problem, using a simple code.
- A Fermi-Hubbard model on a 2×5 lattice and periodic boundary conditions in the lateral direction is considered. We parametrize and compare the sizes of the resulting Hamiltonians, as we employ different codes to save various amounts of qubits. In this way, the trade-off between qubit savings and gate complexity is illustrated (see Table 2.2).

Mapping	En-/Decoding type	Qubits saved	$n(N, K)$	Resulting gates	Origin
Jordan-Wigner | Parity tr.	linear/linear	none	N	length-$O(n)$ Pauli strings	[19, 27]
Bravyi-Kitaev transform	linear/linear	none	N	length-$O(\log n)$ Pauli strings	[26]
Checksum codes	linear/ affine linear	$O(1)$	$N-1$	length-$O(n)$ Pauli strings	[36]
Binary addressing codes	nonlinear/nonlinear	$O(2^{n/K})$	$\log\left(N^K/K!\right)$	$(O(n))$-controlled gates	[36]
Segment codes	linear/nonlinear	$O\left(n/K\right)$	$N/(1+\frac{1}{2K})$	$(O(K))$-controlled gates	[36]

Table 2.1. Overview of mappings presented in this paper, listed by the complexity of their code functions, their qubit savings, qubit requirements (n), properties of the resulting gates and first appearance. Mappings can be compared with respect to the size of plain words (N) and their targeted Hamming weight K. We also refer to different methods that are not listed, as they do not rely on codes in any way [33, 34].

2.3 Encoding the entire Fock space

To illustrate the general use of (possibly nonlinear) encodings to represent fermionic models, let us first briefly generalize how existing mappings can be phrased in terms of linear encodings in the spirit of [27]. Under consideration in representing the dynamics is a mapping for second-quantized Hamiltonians of the form

$$\begin{aligned} H &= \sum_{l=0}^{\infty} \sum_{\substack{\boldsymbol{a}\in[N]^{\otimes l} \\ \boldsymbol{b}\in\mathbb{Z}_2^{\otimes l}}} h_{\boldsymbol{ab}} \prod_{i=1}^{l} (c_{a_i}^{\dagger})^{b_i} (c_{a_i})^{1+b_i \bmod 2} \\ &= \sum_{l} \sum_{\substack{\boldsymbol{a},\boldsymbol{b} \\ \text{with } h_{\boldsymbol{ab}}\neq 0}} \widehat{h}_{\boldsymbol{ab}} \,, \end{aligned} \tag{2.10}$$

where $h_{\boldsymbol{ab}}$ are complex coefficients, chosen in a way as to render H hermitian. For our convenience, we use length-l N-ary vectors $\boldsymbol{a} = (a_1, \dots, a_l)^{\top} \in [N]^{\otimes l}$ to parametrize the orbitals on which a term $\widehat{h}_{\boldsymbol{ab}}$ is acting, and write $[N] = \{1, \dots, N\}$. A similar notation will be employed for binary vectors of length l, with $\boldsymbol{b} = (b_1, \dots, b_l)^{\top} \in \mathbb{Z}_2^{\otimes l}$, $\mathbb{Z}_2 = \{0, 1\}$, deciding whether an operator is a creator or annihilator by the rules $(c_i^{(\dagger)})^1 = c_i^{(\dagger)}$ and $(c_i^{(\dagger)})^0 = 1$.

Every term $\widehat{h}_{\boldsymbol{ab}}$ is a linear operation $\mathcal{F}_N \mapsto \mathcal{F}_N$, with $\mathcal{F}_N$ being the Fock space restricted on N orbitals, the direct sum of all possible anti-symmetrized M-particle Hilbert spaces $\mathcal{H}_N^M$: $\mathcal{F}_N = \bigoplus_{m=0}^{N} \mathcal{H}_N^m$. Conventional mappings transform states of the Fock space $\mathcal{F}_N$ into states on N qubits, carrying over all linear operations as well $\mathcal{L}(\mathcal{F}_N) \mapsto \mathcal{L}((\mathbb{C}^2)^{\otimes N})$.

Before we start presenting conventional transformation schemes, we need to make a few remarks on transformed Hamiltonians and notations pertaining to them. First of all, we identify the set of gates $\{\mathcal{P}, \mathbb{I}\}^{\otimes n} = \{X, Y, Z, \mathbb{I}\}^{\otimes n}$ with the term Pauli strings (on n qubits). The previously mentioned Jordan-Wigner transform, obviously has the power to transform (2.10) into a Hamiltonian that is a weighted sum of Pauli strings on N qubits. General transforms, however, might involve other types of gates. We however have the choice to decompose these into Pauli strings. One might want to do so when using standard techniques for Hamiltonian simulation. In the following, we will denote the correspondence of second quantized operators or states B to their qubit counterparts C by: $B \mathrel{\hat{=}} C$. For convenience, we will also omit identities in Pauli strings and rather introduce qubit labels, e.g. $X \otimes \mathbb{I} \otimes X = X_1 \otimes X_3 = (\bigotimes_{i\in\{1,3\}} X_i)$ and write $\mathbb{I}^{\otimes n} = \mathbb{I}$. A complete table of notations can be found in Section 2.8.

Consider a linear encoding of N fermionic sites into $n = N$ qubits given by a binary matrix A such that

$$|\boldsymbol{\omega}\rangle = |\mathbf{e}\,(\boldsymbol{\nu})\rangle = |A\boldsymbol{\nu}\rangle \hat{=} \left(\prod_{j=1}^{N}(c_j^\dagger)^{\nu_j}\right)|\Theta\rangle \tag{2.11}$$

and A is invertible, i.e. $\left(AA^{-1} \bmod 2\right) = \mathbb{I}$. Note that in this case, the decoding given by $\boldsymbol{\nu} = \mathbf{d}(\boldsymbol{\omega}) = \left(A^{-1}\boldsymbol{\omega}\right)$ is also linear. It is known that any such matrix A, subsequently also yields a mapping of the fermionic creation and annihilation operators to qubit gates [27]. To see how these are constructed, let us start by noting that they must fulfill the properties given in (2.2)-(2.5) and (1.7), which motivates the definition of a parity, a flip and an update set below:

1. $c_{i_m}^{(\dagger)}$ anticommutes with the first $m-1$ operators and thus acquires the phase $(-1)^{m-1}$.

2. A creation operator $c_{i_m}^\dagger$ might be absent (present) in between $c_{i_{m-1}}^\dagger$ and $c_{i_{m+1}}^\dagger$, leading the rightmost operator $c_{i_m}^{(\dagger)}$ to map the entire state to zero since $c_{i_m}|\Theta\rangle = 0 \left(c_{i_m}^\dagger c_{i_m}^\dagger = 0\right)$.

3. Given that the state was not annihilated, the occupation of site i_m has to be changed. This means a creation operator $c_{i_m}^\dagger$ has to be added or removed between $c_{i_{m-1}}^\dagger$ and $c_{i_{m+1}}^\dagger$.

These rules tell us what the transform of an operator $c_j^{(\dagger)}$ has to inflict on a basis state (2.11). In order to implement the phase shift of the first rule, a series of Pauli-Z operators is applied on qubits, whose numbers are in the *parity set* (with respect to $j \in [N]$), $P(j) \subseteq [N]$. Following the second rule we project onto the ± 1 subspace of the Z-string on qubits indexed by another $[N]$ subset, the so-called *flip set* of j, $F(j)$. The *update set* of j, $U(j) \subseteq [N]$ labels the qubits to be flipped completing the third rule using an X-string.

$$(c_j^\dagger)^b (c_j)^{b+1 \bmod 2} \;\hat{=}\; \frac{1}{2}\left(\bigotimes_{k\in U(j)} X_k\right)\left(\mathbb{I}-(-1)^b \bigotimes_{l\in F(j)} Z_l\right)\bigotimes_{m\in P(j)} Z_m\,, \tag{2.12}$$

with $b \in \mathbb{Z}_2$. $P(j)$, $F(j)$ and $U(j)$ depend on the matrices A and A^{-1} as well as the parity matrix R. The latter is a $(N \times N)$ binary matrix which has its lower triangle filled with ones, but not its diagonal. For the matrix entries this means $R_{ij} = \theta_{ij}$, with θ_{ij} as the discrete version of the Heaviside function

$$\theta_{ij} = \begin{cases} 0 & i \leq j \\ 1 & i > j\,, \end{cases} \qquad R = \begin{bmatrix} 0 & & & & \\ 1 & 0 & & & \\ 1 & 1 & 0 & & \\ 1 & 1 & 1 & 0 & \\ \vdots & \vdots & \vdots & \ddots & \ddots \end{bmatrix}. \tag{2.13}$$

The set members are obtained in the following fashion:

1. $P(j)$ contains all column numbers in which the j-th row of matrix RA^{-1} has non-zero entries.

2. $F(j)$ contains the column labels of non-zero entries in the j-th row of A^{-1}.

3. $U(j)$ contains all row numbers in which the j-th column of A has non-zero entries.

Note that this definition of the sets differs from their original appearance in [27, 37], where diagonal elements are not included. In this way, our sets are not disjoint, which leads to Z-cancellations and appearance of Pauli-Y operators, but we have generalized the sets for arbitrary invertible matrices, and provided a pattern for other transforms later.

2.3.1 Jordan-Wigner, Parity and Bravyi-Kitaev transform

As an illustration, we present popular examples of these linear transformations, note again that all of these will have $n = N$. The Jordan-Wigner transform is a special case for $A = \mathbb{I}$, leading to the direct mapping. The operator transform gives $L = O(N)$ Pauli strings as

$$(c_j^\dagger)^b (c_j)^{b+1 \bmod 2} \mathrel{\hat{=}} \frac{1}{2}\left(X_j + i(-1)^b\, Y_j\right) \bigotimes_{m<j} Z_m\,. \tag{2.14}$$

In the parity transform [27], we have $L = O(N)$ X-strings:

$$A^{-1} = \begin{bmatrix} 1 & & & \\ 1 & 1 & & \\ & \ddots & \ddots & \\ & & 1 & 1 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & & & \\ 1 & 1 & & \\ \vdots & \vdots & \ddots & \\ 1 & 1 & \cdots & 1 \end{bmatrix}, \tag{2.15}$$

$$(c_j^\dagger)^b (c_j)^{b+1 \bmod 2} \;\mathrel{\hat{=}}\; \frac{1}{2}\left(Z_{j-1} \otimes X_j - i(-1)^b\, Y_j\right) \bigotimes_{m=j+1}^{N} X_m\,. \tag{2.16}$$

The Bravyi-Kitaev transform [26] is defined by a matrix A [27, 37] that has non-zero entries according to a certain binary tree rule, achieving $L = O(\log N)$.

2.4 Encoding only a subspace

2.4.1 Saving qubits by exploiting symmetries

Our goal is to be able to trade quantum resources, which is done by reducing degrees of freedom by exploiting symmetries. For that purpose, we provide a theoretical foundation to characterize the latter.

Parity, Jordan-Wigner and Bravyi-Kitaev transforms encode all $\mathcal{F}_N$ states and provide mappings for every $\mathcal{L}\left(\mathcal{F}_N\right)$ operator. Unfortunately, they require us to own a N-qubit quantum computer, which might be unnecessary. In fact, the only operator we want to simulate is the Hamiltonian, which usually has certain symmetries. Taking these symmetries into account enables us to perform the same task with $n \leq N$ qubits instead. Symmetries usually divide the $\mathcal{F}_N$ into subspaces, and the idea is to encode only one of those. Let $\mathcal{B}$ be a basis spanning a subspace $\mathrm{span}(\mathcal{B}) \subseteq \mathcal{F}_N$ be associated with a Hamiltonian (2.10), where for every l, $\boldsymbol{a}$, $\boldsymbol{b}$; $\widehat{h}_{\boldsymbol{ab}} : \mathrm{span}(\mathcal{B}) \mapsto \mathrm{span}(\mathcal{B})$. Usually, Hamiltonian symmetries generate many such (distinct) subspaces. Under consideration of

additional information about our problem, like particle number, parity or spin polarization, we select the correct subspace. Note that particle number conservation is by far the most prominent symmetry to take into account. It is generated by Hamiltonians that are linear combinations of products of $c_i^\dagger c_j \,|\, i, j \in [N]$. These Hamiltonians, originating from first principles, only exhibit terms conserving the total particle number; $\widehat{h}_{\boldsymbol{ab}} : \mathcal{H}_N^M \mapsto \mathcal{H}_N^M$. From all the Hilbert spaces $\mathcal{H}_N^M$, one considers the space with the particle number matching the problem description.
These symmetries will be utilized in the next section: we develop a language that allows for encodings $\boldsymbol{e}$ that reduce the length of the binary vectors $\boldsymbol{e}(\boldsymbol{\nu})$ as compared to $\boldsymbol{\nu}$. This means that the state $\boldsymbol{\nu}$ will be encoded in $n \leq N$ qubits, since each bit saved corresponds to a qubit eliminated. As suggested by Bravyi et al. [34], qubit savings can be achieved under the consideration of non-square, invertible matrices A. However, we will see below that using transformations based on nonlinear encodings and decodings $\boldsymbol{d}$ (the inverse transform defined by A^{-1} before), we can eliminate a number of qubits that scales with the system size. For linear codes on the other hand, we find a mere constant saving.

2.4.2 General transforms

We here show how second-quantized operators and states, Hamiltonian symmetries and the fermionic basis $\mathcal{B}$ are fused into a simple description of occupation basis states. While in this section all general ideas are presented, we would like to refer the reader to the appendices for details: to Section 2.7.1 in particular, which holds the proof of the underlying techniques. Fermionic basis states are represented by binary vectors $\boldsymbol{\nu} \in \mathbb{Z}_2^{\otimes N}$, with its components implicating the occupation of the corresponding orbitals. Basis states inside the quantum computer, on the other hand, are represented by binary vectors on a smaller space $\boldsymbol{\omega} \in \mathbb{Z}_2^{\otimes n}$. These vectors are code words of the former $\boldsymbol{\nu}$, where the binary code connecting all $\boldsymbol{\nu}$ and $\boldsymbol{\omega}$ is possibly nonlinear. In the end, an instance of such a code will be sufficient to describe states and operators, in a similar way than the matrix pair $(A,\, A^{-1})$ governs the conventional transforms already presented. We now start by defining such codes and connect them to the state mappings.
Let $\mathrm{span}\,(\mathcal{B})$ be a subspace of $\mathcal{F}_N$, as defined previously. For $n \geq \log|\mathcal{B}|$, we define two binary vector functions $\boldsymbol{d} : \mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2^{\otimes N}$, $\boldsymbol{e} : \mathbb{Z}_2^{\otimes N} \mapsto \mathbb{Z}_2^{\otimes n}$,

where we regard each component $\boldsymbol{d} = (d_1, \dots, d_N)^\top$ as a binary function $d_i : \mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2$. Furthermore we introduce the binary basis set $\mathcal{V} \subseteq \mathbb{Z}_2^{\otimes N}$, with

$$\boldsymbol{\nu} \in \mathcal{V}, \quad \text{only if} \quad \left(\prod_{i=1}^{N} (c_i^\dagger)^{\nu_i} \right) |\Theta\rangle \in \mathcal{B} . \tag{2.17}$$

All elements in $\mathcal{B}$ shall be represented in $\mathcal{V}$. If for all $\boldsymbol{\nu} \in \mathcal{V}$ the binary functions $\boldsymbol{e}$ and $\boldsymbol{d}$ satisfy $\boldsymbol{d}\left(\boldsymbol{e}\left(\boldsymbol{\nu}\right)\right) = \boldsymbol{\nu}$, and for all $\boldsymbol{\omega} \in \mathbb{Z}_2^{\otimes n} : \boldsymbol{d}\left(\boldsymbol{\omega}\right) \in \mathcal{V}$, then we call the two functions encoding and decoding, respectively. An encoding-decoding pair ($\boldsymbol{e}$, $\boldsymbol{d}$) forms a code.

We thus have obtained a general form of encoding, in which qubit states only represent the subspace $\mathrm{span}\left(\mathcal{B}\right)$. The decoding, on the other hand, translates the qubit basis back to the fermionic one:

$$|\boldsymbol{\omega}\rangle = \bigotimes_{j=1}^{n} |\omega_j\rangle \;\hat{=}\; \left(\prod_{i=1}^{N} (c_i^\dagger)^{d_i(\boldsymbol{\omega})} \right) |\Theta\rangle \; . \tag{2.18}$$

We intentionally keep the description of these functions abstract, as the code used might be nonlinear, i.e. it cannot be described with matrices A, A^{-1}. Nonlinearity is thereby predominantly encountered in decoding rather than in encoding functions, as we will see in the examples obtained later.

For any code ($\boldsymbol{e}$, $\boldsymbol{d}$), we will now present the transform of fermionic operators into qubit gates. Before we can do so however, two issues are to be addressed. Firstly, one observes that we cannot hope to find a transformation recipe for a singular fermionic operator $c_j^{(\dagger)}$. The reason for this is that the latter operator changes the occupation of the j-th orbital. As a consequence, a state with the occupation vector $\boldsymbol{\nu}$ is mapped to $\boldsymbol{\nu} + \boldsymbol{u_j}$, where $\boldsymbol{u_j}$ is the unit vector of component j; $\left(u_j\right)_i = \delta_{ij}$. The problem is that since we have trimmed the basis, $\boldsymbol{\nu} + \boldsymbol{u_j}$ will probably not be in $\mathcal{V}$, which means this state is not encoded[1]. The action of $c_j^{(\dagger)}$ is, thus, not defined. We can however obtain a recipe for the non-vanishing Hamiltonian terms $\widehat{h}_{\boldsymbol{ab}}$ as they do not escape the encoded space being $(\mathrm{span}(\mathcal{B}) \mapsto \mathrm{span}(\mathcal{B}))$-operators. Note that this issue is never

[1] 'Unencoded state' is actually a slightly misleading term: when we say a state $\boldsymbol{\lambda} \in \mathbb{Z}_2^{\otimes N}$ is not encoded, we actually mean that it cannot be encoded and correctly decoded, so $\boldsymbol{d}\left(\boldsymbol{e}\left(\boldsymbol{\lambda}\right)\right) \neq \boldsymbol{\lambda}$.

encountered in the conventional transforms, as they encode the entire Fock space.

Secondly, we are yet to introduce a tool to transform fermionic operators into quantum gates. The structure of the latter has to be similar to the linear case, as they mimic the same dynamics as presented in Section 2.3. In general, a gate sequence will commence with some kind of projectors into the subspace with the correct occupation, as well as operators implementing parity phase shifts. The sequence should close with bit flips to update the state. The task is now to determine the form of these operators. The issue boils down to finding operators that extract binary information from qubit states, and map it onto their phase. In other words, we need to find linear operators associated with e.g. the binary function d_j, such that it maps basis states $|\boldsymbol{\omega}\rangle \mapsto (-1)^{d_j(\boldsymbol{\omega})} |\boldsymbol{\omega}\rangle$. In any case, we must recover the case of Pauli strings on their respective sets when considering linear codes. For our example, this means the linear case yields the operator $(\bigotimes_{m \in F(j)} Z_m)$. Using general codes, we are lead to define the extraction superoperation $\mathfrak{X}$, which maps binary functions to quantum gates on n qubits:

$$\mathfrak{X} : \left(\mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2\right) \mapsto \mathcal{L}\left((\mathbb{C}^2)^{\otimes n}\right) . \tag{2.19}$$

The extraction superoperator is defined for all binary vectors $\boldsymbol{\omega} \in \mathbb{Z}_2^{\otimes n}$ and binary functions $f, g : \mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2$ as:

$$\mathfrak{X}[f] \, |\boldsymbol{\omega}\rangle = (-1)^{f(\boldsymbol{\omega})} \, |\boldsymbol{\omega}\rangle$$
$$\text{(Extraction property)} \tag{2.20}$$

$$\mathfrak{X}\left[\boldsymbol{\omega} \mapsto \, f\left(\boldsymbol{\omega}\right) + g\left(\boldsymbol{\omega}\right)\right] = \mathfrak{X}[f] \; \mathfrak{X}[g]$$
$$\text{(Exponentiation identity)} \tag{2.21}$$

$$\mathfrak{X}\left[\boldsymbol{\omega} \mapsto b\right] = (-1)^b \, \mathbb{I} \quad | \, b \in \mathbb{Z}_2$$
$$\text{(Extracting constant functions)} \tag{2.22}$$

$$\mathfrak{X}\left[\boldsymbol{\omega} \mapsto \omega_j\right] = Z_j \quad | \, j \in [n]$$
$$\text{(Extracting linear functions)} \tag{2.23}$$

$$\begin{aligned} &\mathfrak{X}\left[\boldsymbol{\omega} \mapsto \prod_{j \in \mathcal{S}} \omega_j\right] = \mathrm{C}^k\,\textsc{Phase}(i_1, \ldots, i_{k+1}) \\ &\quad \text{with } \mathcal{S} = \{i_s\}_{s=1}^{k+1} \subseteq [n], \;\; k \in [n-1] \\ &\qquad \text{(Extracting nonlinear functions).} \end{aligned} \tag{2.24}$$

Note that the first two properties imply that the operators $\mathfrak{X}[f]$, $\mathfrak{X}[g]$ commute and all operators are diagonal in the computational basis. Given that binary functions have a polynomial form, we are now able to construct operators by extracting every binary function possible, for example

$$\begin{aligned} &\mathfrak{X}[\boldsymbol{\omega} \mapsto 1 + \omega_1 + \omega_1\omega_2] \\ &= \mathfrak{X}\left[\boldsymbol{\omega} \mapsto 1\right]\; \mathfrak{X}\left[\boldsymbol{\omega} \mapsto \omega_1\right]\; \mathfrak{X}\left[\boldsymbol{\omega} \mapsto \omega_1\omega_2\right] && (2.25) \\ &= -Z_1\,\textsc{CPhase}(1,2)\,. && (2.26) \end{aligned}$$

We firstly we have used (2.21) to arrive at (2.25), and then reach (2.26) by applying the properties (2.22)-(2.24) to the respective sub-terms. This might however not be the final Hamiltonian, since the simulation algorithm might require us to reformulate the Hamiltonian as a sum of weighted Pauli strings [38, 39]. In that case, need to decompose all controlled gates. The cost for this decomposition is an increase in the number of Hamiltonian terms, for instance we find $\textsc{CPhase}(i,j) = \frac{1}{2}(\mathbb{I} + Z_i + Z_j - Z_i \otimes Z_j)$. In general, (2.23) and (2.24) can be replaced by an adjusted definition:

$$\mathfrak{X}\left[\boldsymbol{\omega} \mapsto \prod_{j \in \mathcal{S}} \omega_j\right] = \mathbb{I} - 2\prod_{j \in \mathcal{S}} \frac{1}{2}\left(\mathbb{I} - Z_j\right) \quad \Bigg| \; \mathcal{S} \subseteq [n]$$

$$\text{(Extracting non-constant functions).} \tag{2.27}$$

We will be able to define the operator mappings introducing the parity and update functions, $\boldsymbol{p}$ and $\boldsymbol{\varepsilon}^{\boldsymbol{q}}$:

$$\boldsymbol{p} : \mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2^{\otimes N}, \quad p_j\left(\boldsymbol{\omega}\right) = \sum_{i=1}^{j-1} d_i\left(\boldsymbol{\omega}\right)\,, \tag{2.28}$$

$$\begin{aligned} &\boldsymbol{\varepsilon}^{\boldsymbol{q}} : \mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2^{\otimes n}, \quad \text{with } \boldsymbol{q} \in \mathbb{Z}_2^{\otimes N} \\ &\boldsymbol{\varepsilon}^{\boldsymbol{q}}\left(\boldsymbol{\omega}\right) = \boldsymbol{e}\left(\boldsymbol{d}\left(\boldsymbol{\omega}\right) + \boldsymbol{q}\right) + \boldsymbol{\omega}\,. \end{aligned} \tag{2.29}$$

Finally, we have collected all the means to obtain the operator mapping for weight-l operator sequences as they occur in (2.10):

$$\prod_{i=1}^{l}(c_{a_i}^{\dagger})^{b_i}(c_{a_i})^{1+b_i \bmod 2} \;\hat{=}\; \mathcal{U}^{\boldsymbol{a}}\left(\prod_{v=1}^{l-1}\prod_{w=v+1}^{l}(-1)^{\theta_{a_v a_w}}\right)$$
$$\times\prod_{x=1}^{l}\frac{1}{2}\left(\mathbb{I}-\left[\prod_{y=x+1}^{l}(-1)^{\delta_{a_x a_y}}\right](-1)^{b_x}\;\mathfrak{X}\left[d_{a_x}\right]\right)\mathfrak{X}\left[p_{a_x}\right] \tag{2.30}$$

where θ_{ij} is defined in (2.13) and δ_{ij} is the Kronecker delta. In this expression, we find various projectors, parity operators with corrections for occupations that have changed before the update operator is applied. The update operator $\mathcal{U}^{\boldsymbol{a}}$, is characterized by the $\mathbb{Z}_2^{\otimes N}$-vector $\boldsymbol{q}=\sum_{i=1}^{l}\boldsymbol{u}_{\boldsymbol{a_i}}$.

$$\mathcal{U}^{\boldsymbol{a}}=\sum_{\boldsymbol{t}\,\in\,\mathbb{Z}_2^{\otimes n}}\left[\bigotimes_{i=1}^{n}(X_i)^{t_i}\right]\prod_{j=1}^{n}\frac{1}{2}\left(\mathbb{I}+(-1)^{t_j}\;\mathfrak{X}\left[\varepsilon_j^{\boldsymbol{q}}\right]\right). \tag{2.31}$$

This is a problem: when summing over the entire $\mathbb{Z}_2^{\otimes n}$, one has to expect an exponential number of terms. As a remedy, one can arrange the resulting operations into controlled gates, or rely on codes with a linear encoding. If the encoding can be defined using a binary $(n \times N)$-matrix A, $\boldsymbol{e}\left(\boldsymbol{\nu}\right)=A\boldsymbol{\nu}$, the update operator reduces to

$$\mathcal{U}^{\boldsymbol{a}}=\bigotimes_{i=1}^{n}(X_i)^{\sum_j A_{ij}q_j}\;. \tag{2.32}$$

In Section 2.7.1, we show that (2.30)-(2.32) satisfy the conditions (1.7)-(2.5). Note that the update operator is also important for state preparation: let us assume that our qubits are initialized all in their zero state, $(\bigotimes_{i\in[n]}|0\rangle)$, then the fermionic basis state associated with the vector $\boldsymbol{\nu}$ is obtained by applying the update operator $\mathcal{U}^{\boldsymbol{a}}$. Here the vector $\boldsymbol{a}$ contains all occupied orbitals, such that $\boldsymbol{q}=\boldsymbol{\nu}$. Even for nonlinear encodings the state preparation can done with Pauli strings: as the initial state is a product state of all zeros, we can replace operators $\mathfrak{X}[\boldsymbol{\omega}\mapsto\prod_{i\in\mathcal{S}\subseteq[n]}\omega_i]$ by $\mathbb{I}$.

In the following we will turn our attention to the most fruitful symmetry to take into account: particle conservation symmetry. While code families accounting for this symmetry are explored in the next subsection, alternatives to the mapping of entire Hamiltonian terms are discussed for such codes in Section 2.7.2.

2.4.3 Particle number conserving codes

In the following, we will present three types of codes that save qubits by exploiting particle number conservation symmetry, and possibly the conservation of the total spin polarization. Particle number conserving Hamiltonians are highly relevant for quantum chemistry and problems posed from first principles. We therefore set out to find codes in which $\boldsymbol{\nu} \in \mathcal{V}$ have a constant Hamming weight $\mathrm{w_H}\left(\boldsymbol{\nu}\right) = K$. Since the Hamming weight is defined as $\mathrm{w_H}\left(\boldsymbol{\nu}\right) = \sum_m \nu_m$, where the sum is defined without the modulus, it yields the total occupation number for the vectors $\boldsymbol{\nu}$. In order to simulate systems with a fixed particle number, we are thus interested to find codes that implement code words of constant Hamming weight. Note that the fixed Hamming weight K does not necessarily need to coincide with the total particle number M. A code with the such a property might also be interesting for systems with additional symmetries. Most importantly, we have not taken into account the spin multiplicity yet. As the particles in our system are fermions, every spatial site will typically have an even number of spin configurations associated with it. Orbitals with the same spin configurations naturally denote subsets of the total amount of orbitals, much like the suits in a card deck. An absence of magnetic terms as well as spin-orbit interactions leaves the Hamiltonian to conserve the number of particles inside all those suits. Consequently, we can append several constant-weight codes to each other. Each of those subcodes encodes thereby the orbitals inside one suit. In electronic system with only Coulomb interactions for instance, we can use two subcodes ($\boldsymbol{e}^{\diamondsuit}$, $\boldsymbol{d}^{\diamondsuit}$) and ($\boldsymbol{e}^{\spadesuit}$, $\boldsymbol{d}^{\spadesuit}$), to encode all spin-up, and spin-down orbitals, respectively. The global code ($\boldsymbol{e}$, $\boldsymbol{d}$), encoding the entire system, is obtained by appending the subcode functions e.g. $\boldsymbol{d}\left(\boldsymbol{\omega}^{\mathbf{1}} \oplus \boldsymbol{\omega}^{\mathbf{2}}\right) = \boldsymbol{d}^{\diamondsuit}(\boldsymbol{\omega}^{\mathbf{1}}) \oplus \boldsymbol{d}^{\spadesuit}(\boldsymbol{\omega}^{\mathbf{2}})$. Appending codes like this will help us to achieve higher savings at a lower gate cost.

The codes that we now introduce (see also again Table 2.1), fulfill the task of encoding only constant-weight words differently well. The larger $\mathcal{V}$, the less qubits will be eliminated, but we expect the resulting gate sequences to be more simple. Although not just words of that weight are encoded, we treat K as a parameter - the targeted weight.

2.4.3.1 Checksum codes

A slim, constant amount of qubits can be saved with the following $n = N - 1$, affine linear codes. Checksum codes encode all the words with either even or odd Hamming weight. As this corresponds to exactly half of the Fock space, one qubit is eliminated. This means we disregard the last component when we encode $\boldsymbol{\nu}$ into words with one digit less. The decoding function then adds the missing component depending on the parity of the code words. The code for K odd is defined as

$$\boldsymbol{d}(\boldsymbol{\omega}) = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \\ 1 & \cdots & 1 \end{bmatrix} \boldsymbol{\omega} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} , \tag{2.33}$$

$$\boldsymbol{e}(\boldsymbol{\nu}) = \begin{bmatrix} 1 & & & 0 \\ & \ddots & & \vdots \\ & & 1 & 0 \end{bmatrix} \boldsymbol{\nu} \, . \tag{2.34}$$

In the even-K version, the affine vector $\boldsymbol{u_N}$, added in the decoding, is removed. Since encoding and decoding function are both at most affine linear, the extracted operators will all be Pauli strings, with at most a minus sign. The advantage of the checksum codes is that they do not depend on K. They can be used even in cases of smaller saving opportunities, like $K \approx N/2$. We can employ these codes even for Hamiltonians that conserve only the fermion parity. This makes them important for effective descriptions of superconductors [40].

2.4.3.2 Codes with binary addressing

We present a concept for heavily nonlinear codes for large qubit savings, $n = \lceil \log(N^K/K!) \rceil$, [31]. In order to conserve the maximum amount of qubits possible, we choose to encode particle coordinates as binary numbers in $\boldsymbol{\omega}$. To keep it simple, we here consider the example of weight-one binary addressing codes, and refer the reader to Section 2.7.3 for $K > 1$. In $K = 1$, we recognize the qubit savings to be exponential, so consider $N = 2^n$. Encoding and decoding functions are defined by means of the binary enumerator, $\mathrm{bin} : \mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}$, with $\mathrm{bin}\,(\boldsymbol{\omega}) = \sum_{j=1}^{n} 2^{j-1}\omega_j$.

$$d_j\,(\boldsymbol{\omega}) = \prod_{i=1}^{n} \left(\omega_i + 1 + q_i^j\right) \,, \tag{2.35}$$

$$\boldsymbol{e}\,(\boldsymbol{\nu}) = \left[\begin{array}{c|c|c|c} \boldsymbol{q^1} & \boldsymbol{q^2} & \quad\cdots\quad & \boldsymbol{q^{2^n}} \end{array}\right] \boldsymbol{\nu}\,, \tag{2.36}$$

where $\boldsymbol{q^j} \in \mathbb{Z}_2^{\otimes n}$ is implicitly defined by $\mathrm{bin}(\boldsymbol{q^j}) + 1 = j$. An input $\boldsymbol{\omega}$ will by construction render only the j-th component of (2.35) non-zero, when $\boldsymbol{q^j} = \boldsymbol{\omega}$.

The exponential qubit savings come at a high cost: the product over each component of $\boldsymbol{\omega}$ implies multi-controlled gates on the entire register. This is likely to cause connectivity problems. Note that decomposing the controlled gates will in general be practically prohibited by the sheer amount of resulting terms. On top of those drawbacks, we also expect the encoding function to be nonlinear for $K > 1$.

2.4.3.3 Segment codes

We introduce a type of scaleable $n = \lceil N/(1 + \frac{1}{2K})\rceil$ codes to eliminate a linear amount of qubits. The idea of segment codes is to cut the vectors $\boldsymbol{\nu}$ into smaller, constant-size vectors $\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}} \in \mathbb{Z}_2^{\otimes \widehat{N}}$, such that $\boldsymbol{\nu} = \bigoplus_i \widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$. Each such segment $\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$ is encoded by a subcode. Although we have introduced the concept already, this segmentation is independent from our treatment of spin 'suits'. In order to construct a weight-K global code, we append several instances of the same subcode. Each of these subcodes codes is defined on $\widehat{n}$ qubits, encoding $\widehat{N} = \widehat{n} + 1$ orbitals. We deliberately have chosen to only save one qubit per segment in order to keep the segment size $\widehat{N}(K)$ small.

We now turn our attention to the construction of these segment codes. As shown in Section 2.7.4, the segment sizes can be set to $\widehat{n} = 2K$ and $\widehat{N} = 2K + 1$. As the global code is supposed to encode all $\boldsymbol{\nu} \in \mathbb{Z}_2^{\otimes N}$ with Hamming weight K, each segment must encode all vectors from Hamming weight zero up to weight K. In this way, we guarantee that the encoded space contains the relevant, weight-K subspace. This construction follows from the idea that each block contains equal or less than K

particles, but might as well be empty. For each segment, the following de- and encoding functions are found for $\widehat{\boldsymbol{\omega}} \in \mathbb{Z}_2^{\otimes \widehat{n}}$, $\widehat{\boldsymbol{\nu}} \in \mathbb{Z}_2^{\otimes \widehat{N}}$:

$$\widehat{\boldsymbol{d}}\left(\widehat{\boldsymbol{\omega}}\right) = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \\ 0 & \dots & 0 \end{bmatrix} \widehat{\boldsymbol{\omega}} \; + \; f\left(\widehat{\boldsymbol{\omega}}\right) \begin{pmatrix} 1 \\ \vdots \\ \vdots \\ 1 \end{pmatrix} \tag{2.37}$$

$$\widehat{\boldsymbol{e}}\left(\widehat{\boldsymbol{\nu}}\right) = \begin{bmatrix} 1 & & 1 \\ & \ddots & \vdots \\ & 1 & 1 \end{bmatrix} \widehat{\boldsymbol{\nu}}\,, \tag{2.38}$$

where $f : \mathbb{Z}_2^{\otimes \widehat{n}} \mapsto \mathbb{Z}_2$ is a binary switch. The switch is the source of nonlinearity in these codes. On an input $\widehat{\boldsymbol{\omega}}$ with $\mathrm{w_H}\left(\widehat{\boldsymbol{\omega}}\right) > K$, it yields one, and zero otherwise.
There is just one problem: segment codes are not suitable for particle-number conserving Hamiltonians, according to the definition of the basis $\mathcal{B}$, that we would have for segment codes. The reason for this is that we have not encoded all states with $\mathrm{w_H}\left(\boldsymbol{\nu}\right) > K$. In this way, Hamiltonian terms $\widehat{h}_{\boldsymbol{ab}}$ that exchange occupation numbers between two segments, can map into unencoded space. We can, however, adjust these terms, such that they only act non-destructively on states with at most K particles between the involved segment. This does not change the model, but aligns the Hamiltonian with the necessary condition that we have on $\mathcal{B}$, $\widehat{h}_{\boldsymbol{ab}} : \mathrm{span}(\mathcal{B}) \mapsto \mathrm{span}(\mathcal{B})$. This is discussed in detail Section 2.7.4, where we also provide an explicit description of the binary switch mentioned earlier.

Using segment codes, the operator transforms will have multi-controlled gates as well: the binary switch is nonlinear. However, gates are controlled on at most an entire segment, which means there is no gate that acts on more than $2K$ qubits. This an improvement in gate locality, as compared to binary addressing codes.

2.5 Examples

2.5.1 Hydrogen molecule

In this subsection, we will demonstrate the Hamiltonian transformation on a simple problem. Choosing a standard example, we draw comparison with other methods for qubit reduction. As one of the simplest problems, the minimal electronic structure of the hydrogen molecule has been studied extensively for quantum simulation [23, 38] already. We describe the system as two electrons on 2 spatial sites. Because of the spin-multiplicity, we require 4 qubits to simulate the Hamiltonian in conventional ways. Using the particle conservation symmetry of the Hamiltonian, this number can be reduced. The Hamiltonian also lacks terms that mix spin-up and -down states, with the total spin polarization known to be zero in the ground state. Taking into account these symmetries, one finds a total of 4 fermionic basis states:
$\mathcal{V} = \{(0,1,0,1)\,,(0,1,1,0)\,,(1,0,0,1)\,,(1,0,1,0)\}$. These can be encoded into two qubits by appending two instances of a ($N = 2$, $n = 1$, $K = 1$)-code. The global code is defined as :

$$\boldsymbol{d}\,(\boldsymbol{\omega}) = \begin{bmatrix} 1 & \\ 1 & \\ & 1 \\ & 1 \end{bmatrix} \boldsymbol{\omega} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \tag{2.39}$$

$$\boldsymbol{e}\,(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{\nu}\,. \tag{2.40}$$

The physical Hamiltonian,

$$\begin{aligned} H = & - h_{11}\left(c_1^\dagger c_1 + c_3^\dagger c_3\right) - h_{22}\left(c_2^\dagger c_2 + c_4^\dagger c_4\right) \\ & + h_{1331}\ c_1^\dagger c_3^\dagger c_3 c_1 + h_{2442}\ c_2^\dagger c_4^\dagger c_4 c_2 \\ & + h_{1221}\left(c_1^\dagger c_4^\dagger c_4 c_1 + c_3^\dagger c_2^\dagger c_2 c_3\right) \\ & + (h_{1221} - h_{1212})\left(c_1^\dagger c_2^\dagger c_2 c_1 + c_3^\dagger c_4^\dagger c_4 c_3\right) \\ & + h_{1212}\left(c_1^\dagger c_4^\dagger c_3 c_2 + c_2^\dagger c_3^\dagger c_4 c_1\right) \\ & + h_{1212}\left(c_1^\dagger c_3^\dagger c_4 c_2 + c_2^\dagger c_4^\dagger c_3 c_1\right)\ , \end{aligned} \tag{2.41}$$

is transformed into the qubit Hamiltonian

$$g_1\,\mathbb{I} + g_2\,X_1 \otimes X_2 + g_3\,Z_1 + g_4\,Z_2 + g_5\,Z_1 \otimes Z_2\,. \tag{2.42}$$

The real coefficients g_i are formed by the coefficients h_{ijkl} of (2.41). After performing the transformation, we find

$$\begin{aligned} g_1 &= -h_{11} - h_{22} + \frac{1}{2}h_{1221} + \frac{1}{4}h_{1331} + \frac{1}{4}h_{2442} && (2.43)\\ g_2 &= h_{1212} && (2.44)\\ g_3 &= g_4 = \frac{1}{2}h_{11} - \frac{1}{2}h_{22} + -\frac{1}{4}h_{1331} + \frac{1}{4}h_{2442} && (2.45)\\ g_5 &= -\frac{1}{2}h_{1221} + \frac{1}{4}h_{1331} + \frac{1}{4}h_{2442}\,. && (2.46) \end{aligned}$$

In previous works, conventional transforms have been applied to that problem Hamiltonian. Afterwards, the resulting 4-qubit-Hamiltonian has been reduced by hand in some way. In [41], the actions on two qubits are replaced with their expectation values after inspection of the Hamiltonian. In [33], on the other hand, the Hamiltonian is reduced to two qubits in a systematic fashion. Finally, the case is revisited in [34], where the problem is reduced below the combinatorial limit to one qubit. The latter two attempts have used Jordan-Wigner, the former the Bravyi-Kitaev transform first.

2.5.2 Fermi-Hubbard model

We present another example to illustrate the trade-off between qubit number and gate cost as well as circuit depth. For that purpose, we consider a simple toy Hamiltonian and demonstrate that a reduction of qubit requirements is theoretically possible. Although we do not want to claim that this scenario is realistic, we present a simple cost model with it, that hints the potential up-scaling of circuit depth and simulation cost, as the number of qubits decreases: we therefore consider the total sum of Pauli lengths of every term, which gives us an idea of the number of two-qubit gates required, and the number of Hamiltonian terms, as we decompose controlled gates (2.27), which should give us an idea of possible T-gate requirements and simulation depth. Let us start now to describe the model. We consider a small lattice with periodic boundary conditions in the lateral direction. The system shall contain 10 spatial sites, doubled by the spin-multiplicity. The problem Hamiltonian is

$$\begin{aligned} H = &- t \sum_{\langle i,j \rangle \in E} \left(c_i^\dagger c_j + c_j^\dagger c_i \right) \\ &+ U \sum_{j=1}^{10} c_j^\dagger c_j \, c_{10+j}^\dagger c_{10+j} \, , \end{aligned} \tag{2.47}$$

with its real coefficients t, U. It exhibits hopping terms along the edges E of the graph in Figure 2.1. The sketch on the left of this figure shows the connection graph of the first 10 orbitals. The other 10 orbitals are connected in the same fashion, and each such site is interacting with its counterpart from the other graph. We aim to populate this model with four fermions, where the total spin polarization is zero. Two conventional transforms and two transforms based on our codes are compared by the amount of qubits necessary, as well as the size of the transformed Hamiltonian. Note that besides eigenenergies, one might also be interested in obtaining the values of correlation functions, e.g. $\langle c_i^\dagger c_j \rangle$, which is done by measuring (qubit) operators obtained with the transform (2.47). The only difference is that if a correlator maps into unencoded space, it is to be set to zero. As benchmarks, we decompose controlled gates and count the number of resulting Pauli strings. The sum of their total weight constitutes the gate count. Having these two disconnected graphs is an invitation to us to append two codes acting on sites $1 - 10$ and $11 - 20$ respectively. For this example, we consider the following codes:

1. Jordan-Wigner and Bravyi-Kitaev transform: for comparison, we employ these conventional transforms on our system, with which we do not save qubits. The resulting terms are best obtained by the transforming every fermion operator in (2.47) by (2.12), where the flip, parity and update sets, $F(j)$, $P(j)$, $U(j)$ are determined by the choice of matrices A and A^{-1}, which are binary-tree matrices in the case of the Bravyi-Kitev transform, and identity matrices for the Jordan-Wigner transform.

2. Checksum code $\oplus$ checksum code: knowing that the particle number is conserved, and that spin cannot be flipped, we are free to save 2 qubits in constraining the parity of both, spin-up and -down particles, alike. This is done in appending two (N=10) checksum codes, where each that acts on only spin-up (spin-down) orbitals,

so indices 1 to 10 (11 to 20). The code resulting from appending two even checksum codes is linear, and encoding and decoding function feature the matrices A, A^{-1} as

$$\left[\begin{array}{ccc|ccc} 1 & & 0 & & & \\ & \ddots & \vdots & & & \\ & 1 & 0 & & & \\ \hline & & & 1 & & 0 \\ & & & & \ddots & \vdots \\ & & & & 1 & 0 \end{array}\right] , \left[\begin{array}{ccc|ccc} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ 1 & \cdots & 1 & & & \\ \hline & & & 1 & & \\ & & & & \ddots & \\ & & & & & 1 \\ & & & 1 & \cdots & 1 \end{array}\right] . \tag{2.48}$$

However, as not the entire Fock-space is encoded, we need to perform the operator transform according to (2.30), where the update operator is defined by (2.32), where A refers here to the first matrix.

3. Segment code $\oplus$ segment code: Knowing the particle number in one 'spin suite' to be 2, we can for both, spin-up and -down orbitals, append two $K = 2$ segment codes to each other. This equals a total of 4 segment codes, saving 4 qubits. The resulting global code $(\boldsymbol{e},\, \boldsymbol{d})$ is defined by

$$\boldsymbol{e}\left(\bigoplus_{i=1}^{4} \widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}\right) = \bigoplus_{i=1}^{4} \widehat{\boldsymbol{e}}(\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}), \tag{2.49}$$

$$\boldsymbol{d}\left(\boldsymbol{\omega}\right) = \left(\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & & 1 \\ & \ddots & \vdots \\ & 1 & 1 \end{bmatrix}\right) \boldsymbol{\omega} , \tag{2.50}$$

where $\widehat{\boldsymbol{e}}$ are the encodings of the subcodes (2.38), and $\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$ are occupations on the segments of the total orbital vector $\boldsymbol{\nu} = \bigoplus_i \widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$. These segments are formed as suggested by the right-hand side of Figure 2.1. For details on the decoding functions and Hamiltonian adjustments, please consider Section 2.7.4. The Hamiltonian transform is in the end carried out again by (2.30) and (2.32).

4. Checksum code $\oplus$ segment code: a compromise between the above, in which the spin-up orbitals are transformed via a checksum code, and the spin-down orbitals are transformed via two segment codes. The global code used for the Hamiltonian transformation is the

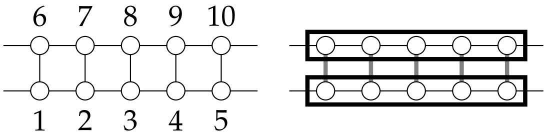


Figure 2.1. Left: illustration of the Fermi-Hubbard model considered. Lines between two sites, like 1 and 2, indicate the appearance of the term $t(c_1^\dagger c_2 + c_2^\dagger c_1)$ in the Hamiltonian (2.47). Periodic boundary conditions link sites 1 and 5 as well as 6 and 10. Sites 11-20 follow the same graph. Right: segmenting of the system; the two blocks are infringed. The gray links are to be adjusted.

appendage of an (even-weight, $N = 10$) checksum code and two $(K = 2)$ segment codes, including Hamiltonian adjustments on the spin-down orbitals.

Note that from the combinatorial perspective, we could encode the problem with 11 qubits. However, if we append two $K = 2$ binary addressing codes to each other, the resulting Hamiltonian is on 14 qubits already. The problem is that the resulting Hamiltonian for this case cannot be expressed with decomposed controlled gates due to the high number of resulting terms.

Indeed, Table 2.2 suggests that decomposing the controlling gates might easily lead to very large Hamiltonians with a multitude of very small terms. The gate decomposition appears therefore undesirable. We in general recommend to rather decompose large controlled gates as shown in [42]. However, one also notices that an elimination of up to two qubits comes at a low cost: the amount of gates is not higher than in the Bravyi-Kitaev transform. As soon as we employ segment codes on the other hand, the Hamiltonian complexity rises with the amount of qubits eliminated.

2.6 Conclusion

In this chapter, we have introduced new methods to reduce the number of qubits required for simulating fermionic systems in second quantization. We see the virtue of the introduced concepts in the fact that it takes into account symmetries on a simple but non-abstract level. We merely

Mapping	Qubits	Weight	Terms
Jordan-Wigner transform	20	232	74
Bravyi-Kitaev transform	20	278	74
Checksum code $\oplus$ Checksum code	18	260	74
Checksum code $\oplus$ Segment code	17	4425	876
Segment code $\oplus$ Segment code	16	9366	1838

Table 2.2. Relaxing the qubit requirements for the Hamiltonian (2.47), where various mappings trade different amounts of qubits. The notation $\oplus$ is used as two codes for different graphs are appended. We compare different mappings by the amount of qubits. We make comparrisons by the number of Hamiltonian terms and the total weight of the resulting Pauli strings.

concern ourselves with objects as simple as binary vectors, but attribute the physical interpretation of orbital occupations to them. At this level, the mentioned symmetries are easy to apply and exploit. The accounting for the complicated antisymmetrization of the many-body wave function on the other hand is done in the fermionic operators, which to transform we have provided recipes for. In these operator transforms we see room for improvement: we for instance lack a proper gate composition for update operators of nonlinear encodings at this point. We on the other hand have the extraction superoperator $\mathfrak{X}$ return only conventional (multi)-controlled phase gates. Nonlinear codes would on the other hand benefit from a gate set that includes gates with negative control, i.e. with the (-1) eigenvalue conditioned on $|0\rangle$ eigenspaces of certain qubits involved. We consider our work to be relevant for quantum simulation with near-term devices, with a limited number of qubits at disposal. Remarks about asymptotic scaling are thus missing in this work, but would be interesting. Also, we have centered our investigations around quantum computers with qubits. The idea behind the generalized operator transforms, however, can possibly be adapted to multi-level systems (qudits). The operator transforms of segment and binary addressing codes, for instance, might simplify in such a setup, if generalized Pauli operators are available in some form.

Apart from the codes presented, we have laid the foundation for the reader to invent their own. For that purpose, we have added the functionality of defining and using binary code transforms (with linear encoding functions $\boldsymbol{e}$) to the *OpenFermion* software package [11].

2.7 Supplement

2.7.1 General operator mappings

The goal of this section is to verify that the fermionic mode is accurately represented by our qubit system. This is divided into three steps: step one is to analyze the action of Hamiltonian terms on the fermionic basis. In the second step, we verify parity and projector parts of (2.30) to work like the original operators in step one, disregarding the occupational update for a moment. Conditions for this state update are subsequently derived. The update operator (2.31) is shown to fulfill these conditions in the third step, thus concluding the proof.

2.7.1.1 Hamiltonian dynamics

In order to verify that the gate sequences (2.30) are mimicking the Hamiltonian dynamics adequately, we verify that the resulting terms have the same effect on the Hamiltonian basis. This is done on the level of second quantization with respect to the notation (2.17): no transition into a qubit system is made. This step serves the sole purpose to quantify the effect of the Hamiltonian terms on the states. To that end, we begin by studying the effect of a singular fermionic operator $c_j^{(\dagger)}$ on a pure state, before considering an entire term $\widehat{h}_{\boldsymbol{ab}}$ on a state in $\mathcal{B}$. As a preliminary, we note that (2.2)-(2.5) follow directly from (1.7), when considering that

$$c_j c_j = c_j^\dagger c_j^\dagger = c_j \left|\Theta\right\rangle = 0\,. \tag{2.51}$$

The relations (2.2)-(2.5) indicate how singular operators act on pure states in general. We now become more specific and apply these rules to a state $(\prod_i (c_i^\dagger)^{\nu_i}) \left|\Theta\right\rangle$, that is not necessarily in $\mathcal{B}$, but is described by an occupation vector $\boldsymbol{\nu} \in \mathbb{Z}_2^{\otimes N}$. The effect of an annihilation operator on such a state is considered first:

$$c_j \left[\prod_{i=1}^{N} \left(c_i^\dagger\right)^{\nu_i}\right] |\Theta\rangle = \left[\prod_{i<j} \left(-c_i^\dagger\right)^{\nu_i}\right] \; c_j \left(c_j^\dagger\right)^{\nu_j} \; \left[\prod_{k>j} \left(c_k^\dagger\right)^{\nu_k}\right] |\Theta\rangle \tag{2.52}$$

$$= \left[\prod_{i<j} \left(-c_i^\dagger\right)^{\nu_i}\right] \; \frac{1}{2}\left[1-(-1)^{\nu_j}\right] \; \left[\prod_{k>j} \left(c_k^\dagger\right)^{\nu_k}\right] |\Theta\rangle \tag{2.53}$$

$$= \left[\prod_{i<j} (-1)^{\nu_i}\right] \; \frac{1}{2}\left[1-(-1)^{\nu_j}\right] \; \left[\prod_{k=1}^{N} \left(c_k^\dagger\right)^{\nu_k+\delta_{jk} \bmod 2}\right] |\Theta\rangle \tag{2.54}$$

A short explanation on what has happened: in (2.52), c_j has anticommuted with all creation operator $c_i^\dagger$ that have indexes $i < j$. Depending on the component ν_j, a creation operator $c_j^\dagger$ might now be to the right of the annihilator c_j. If the creation operator is not encountered, we may continue the anticommutations of c_j until it meets the vacuum and annihilates the state by $c_j |\Theta\rangle = 0$. Using the anticommutation relations (1.7), we therefore replace $c_j(c_j^\dagger)^{\nu_j}$ with $\frac{1}{2}\left[1-(-1)^{\nu_j}\right]$ when going from (2.52) to (2.53). Finally, the terms are rearranged in (2.54): conditional sign changes of the anticommutations are factored out of the new state with an occupation that is now described by the binary vector $(\boldsymbol{\nu} + \boldsymbol{u_j})$ rather than $\boldsymbol{\nu}$. When considering to apply a creation operator $c_j^\dagger$ on the former state, the result is similar. Alone at step (2.53), we have to replace $c_j^\dagger(c_j^\dagger)^{\nu_j}$ by $\frac{1}{2}\left[1+(-1)^{\nu_j}\right]$ instead, as now the case of appearance of the creation operator leads to annihilation: $c_j^\dagger c_j^\dagger = 0$. We thus find

$$c_j^\dagger \left[\prod_{i=1}^{N} \left(c_i^\dagger\right)^{\nu_j}\right] |\Theta\rangle = \left[\prod_{i<j} (-1)^{\nu_i}\right] \; \frac{1}{2}\left[1+(-1)^{\nu_j}\right] \; \left[\prod_{k=1}^{N} \left(c_k^\dagger\right)^{\nu_k+\delta_{jk} \bmod 2}\right] |\Theta\rangle . \tag{2.55}$$

We now turn our attention to the actual goal, the effect that a Hamiltonian term from (2.10) has on a state in $\mathcal{B}$ (this means its occupation vector $\boldsymbol{\nu}$ is in $\mathcal{V}$). We therefore consider a generic operator sequence $\prod_{i=1}^{l}(c_{a_i}^\dagger)^{b_i}(c_{a_i})^{1+b_i \bmod 2}$, parametrized by some N-ary vector $\boldsymbol{a} \in [N]^{\otimes l}$ and a binary vector $\boldsymbol{b} \in \mathbb{Z}_2^{\otimes l}$, for some length l. With (2.54) and (2.55), we now have the means to consider the effect such a sequence of annihilation and creation operators. The two relations will be repeatedly utilized in an inductive procedure, as every single operator $(c_{a_i}^\dagger)^{b_i}(c_{a_i})^{1+b_i \bmod 2}$

of $\prod_{i=1}^{l}(c^\dagger_{a_i})^{b_i}(c_{a_i})^{1+b_i \bmod 2}$ will act on a basis state, one after another. The state's occupation is updated after every such operation. For convenience, we define:

$$\boldsymbol{\nu}^{(i)} \in \mathbb{Z}_2^{\otimes N} \quad \Big| \quad i \in \{0,\, \ldots,\, l\} \tag{2.56}$$

$$\boldsymbol{\nu}^{(l)} = \boldsymbol{\nu} \in \mathcal{V} \tag{2.57}$$

$$\boldsymbol{\nu}^{(i-1)} = \boldsymbol{\nu}^{(i)} + \boldsymbol{u}_{\boldsymbol{a_i}} \ . \tag{2.58}$$

Now, the procedure starts:

$$\left[\prod_{i=1}^{l}(c^\dagger_{a_i})^{b_i}(c_{a_i})^{1+b_i \bmod 2}\right] \left[\prod_{k=1}^{N}(c^\dagger_k)^{\nu_k}\right]|\Theta\rangle \tag{2.59}$$

$$= \left[\prod_{i=1}^{l-1}(c^\dagger_{a_i})^{b_i}(c_{a_i})^{1+b_i \bmod 2}\right] \frac{1}{2}\left[1-(-1)^{b_l}\,(-1)^{\nu_{a_l}}\right]$$

$$\times\,(-1)^{\sum_{j<a_l}\nu_j}\left[\prod_{k=1}^{N}\left(c^\dagger_k\right)^{\nu_k+\delta_{a_l k} \bmod 2}\right]|\Theta\rangle \tag{2.60}$$

$$= \left[\frac{1}{2}\left[1-(-1)^{b_l}\,(-1)^{\nu^{(l)}_{a_l}}\right](-1)^{\sum_{j<a_l}\nu^{(l)}_j}\right]$$

$$\times\left[\prod_{i=1}^{l-1}\left(c^\dagger_{a_i}\right)^{b_i}\left(c_{a_i}\right)^{1+b_i \bmod 2}\right]\left[\prod_{k=1}^{N}(c^\dagger_k)^{\nu^{(l-1)}_k}\right]|\Theta\rangle \tag{2.61}$$

$$= \left[\prod_{i=1}^{l}\underbrace{\frac{1}{2}\left[1-(-1)^{b_i}\,(-1)^{\nu^{(i)}_{a_i}}\right]}_{\text{projector eigenvalues}}\underbrace{(-1)^{\sum_{j<a_i}\nu^{(i)}_j}}_{\text{parity signs}}\right]\underbrace{\left[\prod_{k=1}^{N}(c^\dagger_k)^{\nu^{(0)}_k}\right]|\Theta\rangle}_{\text{updated state}} \tag{2.62}$$

We again explain what has happened: first, the rightmost operator, which is either c_{a_l} or $c^\dagger_{a_l}$ depending on the parameter b_l, acts on the state according to either (2.54) or (2.55). We therefore combine the two relations for the absorption of this operator $(c^\dagger_{a_l})^{b_l}(c_{a_l})^{1+b_l \bmod 2}$ in (2.60). In the same fashion, all the remaining operators of the sequence are one-after-another absorbed into the state. The new state is described by the vector $\boldsymbol{\nu}^{(l-1)}$ after the update. And the cycle begins anew with $(c^\dagger_{a_{l-1}})^{b_{l-1}}(c_{a_{l-1}})^{1+b_{l-1} \bmod 2}$. From (2.61) on, we use the notations (2.56)-(2.58) to describe partially updated occupations. By the end of this iteration, the occupation of the state is changed to $\boldsymbol{\nu}^{(0)} = \boldsymbol{\nu} + \boldsymbol{q}$, with the total change $\boldsymbol{q} = \sum_i \boldsymbol{u}_{\boldsymbol{a_i}}$. Also, the coefficients of (2.62) take into account sign changes from anticommutations ("parity signs" in (2.62)) and the eigenvalues of the applied projections. In its entirety, (2.62) denotes the resulting state, and is the main ingredient for the next step.

2.7.1.2 Parity operators and projectors

We are given the operator transform (2.30) and the state transform (2.18). We want to show the that the fermion system is adequately simulated, which means to show that the effect (2.62) is replicated by (2.30) acting on $|\boldsymbol{e}(\boldsymbol{\nu})\rangle$. This is the goal of the next two steps. We start by evaluating the application of (2.30) on that state, up to the update operator $\mathcal{U}^{a}$. This means that the operators applied implement two things only: the parity signs of (2.62), and the projection onto the correct occupational state. Note that these parity operators and projectors are applied before the update operator in (2.30):

$$\begin{aligned}
&\overbrace{\mathcal{U}^{\boldsymbol{a}}}^{\textbf{update operator}}\ \overbrace{\left(\prod_{v=1}^{l-1}\prod_{w=v+1}^{l}(-1)^{\theta_{a_v a_w}}\right)}^{\textbf{parity signs}}\\
&\times\prod_{x=1}^{l}\frac{1}{2}\underbrace{\left(\mathbb{I}-\left[\prod_{y=x+1}^{l}(-1)^{\delta_{a_x a_y}}\right](-1)^{b_x}\,\mathfrak{X}\left[d_{a_x}\right]\right)}_{\textbf{projectors}}\ \underbrace{\mathfrak{X}\left[p_{a_x}\right]}_{\textbf{parity operators}}\ . \qquad (2.63)
\end{aligned}$$

We now commence our evaluation:

$$\begin{aligned}
&\mathcal{U}^{\boldsymbol{a}}\left[\left(\prod_{v=1}^{l-1}\prod_{w=v+1}^{l}(-1)^{\theta_{a_v a_w}}\right)\right.\\
&\quad\left.\prod_{x=1}^{l}\frac{1}{2}\left(\mathbb{I}-\left[\prod_{y=x+1}^{l}(-1)^{\delta_{a_x a_y}}\right](-1)^{b_x}\,\mathfrak{X}\left[d_{a_x}\right]\right)\mathfrak{X}\left[p_{a_x}\right]\right]|\boldsymbol{e}\left(\boldsymbol{\nu}\right)\rangle \qquad (2.64)
\end{aligned}$$

$$\begin{aligned}
=\ &\mathcal{U}^{\boldsymbol{a}}\left[\left(\prod_{v=1}^{l-1}\prod_{w=v+1}^{l}(-1)^{\theta_{a_v a_w}}\right)\right.\\
&\prod_{x=1}^{l}\frac{1}{2}\left(1-\left[\prod_{y=x+1}^{l}(-1)^{\delta_{a_x a_y}}\right](-1)^{b_x}\,(-1)^{d_{a_x}(\boldsymbol{e}(\boldsymbol{\nu}))}\right)\left.(-1)^{p_{a_x}(\boldsymbol{e}(\boldsymbol{\nu}))}\right]|\boldsymbol{e}\left(\boldsymbol{\nu}\right)\rangle \\
&\qquad (2.65)
\end{aligned}$$

$$= \mathcal{U}^{\boldsymbol{a}} \left[\left(\prod_{v=1}^{l-1} \prod_{w=v+1}^{l} (-1)^{\theta_{a_v a_w}} \right) \right.$$

$$\left. \prod_{x=1}^{l} \frac{1}{2} \left(1 - \left[\prod_{y=x+1}^{l} (-1)^{\delta_{a_x a_y}} \right] (-1)^{b_x} \, (-1)^{\nu_{a_x}} \right) (-1)^{\sum_{j<a_x} \nu_j} \right] \left| \boldsymbol{e}\left(\boldsymbol{\nu}\right) \right\rangle \qquad (2.66)$$

$$= \mathcal{U}^{\boldsymbol{a}} \left[\prod_{x=1}^{l} \frac{1}{2} \left(1 - (-1)^{b_x} \, (-1)^{\nu_{a_x} + \sum_{y=x+1}^{l} \delta_{a_x a_y}} \right) \right.$$

$$\left. (-1)^{\sum_{j<a_x} \nu_j + \sum_{y=x+1}^{l} \theta_{a_x a_y}} \right] \quad \left| \boldsymbol{e}\left(\boldsymbol{\nu}\right) \right\rangle \qquad (2.67)$$

$$= \quad \left[\prod_{x=1}^{l} \frac{1}{2} \left(1 - (-1)^{b_x} \, (-1)^{\nu_{a_x}^{(x)}} \right) (-1)^{\sum_{j<a_x} \nu_j^{(x)}} \right] \quad \mathcal{U}^{\boldsymbol{a}} \, \left| \boldsymbol{e}\left(\boldsymbol{\nu}\right) \right\rangle \, . \qquad (2.68)$$

Let us describe what has happened: in (2.65), the extraction property (2.20) is used, and we arrive at (2.66) after using the property $\boldsymbol{d}\left(\boldsymbol{e}\left(\boldsymbol{\nu}\right)\right) = \boldsymbol{\nu}$ and the definition of the parity function. From there we go to (2.67) when we merge the two products and perform rearrangements that make it easy to cast all delta and theta functions into the components of the partially updated occupations $\boldsymbol{\nu}^{(i)}$, (2.68).

Comparing (2.68) to (2.62), we notice to have successfully mimicked the same sign changes and and projections, as the coefficients in both relations match. Now it is only left to show that the state update is executed correctly. Naively, one would think that we would need to show that

$$\mathcal{U}^{\boldsymbol{a}} \, \left| \boldsymbol{e}\left(\boldsymbol{\nu}\right) \right\rangle \hat{=} \left[\prod_{k=1}^{N} \left(c_k^{\dagger} \right)^{\nu_k^{(0)}} \right] \left| \Theta \right\rangle \, , \qquad (2.69)$$

but this is too strong a statement. It is in fact sufficient to demand

$$\mathcal{U}^{\boldsymbol{a}} \left| \boldsymbol{e}\left(\boldsymbol{\nu}\right) \right\rangle \; = \; \left| \boldsymbol{e}\left(\boldsymbol{\nu}^{(0)}\right) \right\rangle \; = \; \left| \boldsymbol{e}\left(\boldsymbol{\nu} + \boldsymbol{q}\right) \right\rangle \, . \qquad (2.70)$$

For $\boldsymbol{\nu}^{(0)} \in \mathcal{V}$, (2.69) and (2.70) is equivalent. However, it might be the case that $\boldsymbol{\nu}^{(0)} \notin \mathcal{V}$, so $\boldsymbol{\nu}^{(0)}$ is not encoded. This mean that (2.69) is not fulfilled, since $\boldsymbol{d}(\boldsymbol{e}(\boldsymbol{\nu}^{(0)})) \neq \boldsymbol{\nu}^{(0)}$. It is however not necessary to include $\boldsymbol{\nu}^{(0)}$ in the encoding, as for $\boldsymbol{\nu}^{(0)} \notin \mathcal{V}$, the state will vanish anyways: we know from $\widehat{h}_{\boldsymbol{ab}} : \mathrm{span}(\mathcal{B}) \mapsto \mathrm{span}(\mathcal{B})$, that in this case $\widehat{h}_{\boldsymbol{ab}}$ must act destructively on that basis state, $\widehat{h}_{\boldsymbol{ab}}\,(\prod_k (c_k^\dagger)^{\nu_k})\,|\Theta\rangle = 0$. This detail is implemented by the projector part of the transformed sequence (2.30). These projectors are, as we have just shown, working faithfully like (2.62), for the transformed sequence acting on every $|\boldsymbol{\nu}\rangle$ with $\boldsymbol{\nu} \in \mathcal{V}$. Hence (2.70) is a sufficient condition for the updated state. The proof is completed once we have verified that (2.70) is satisfied with the update operator defined as in (2.31). This is done during the next step.

2.7.1.3 Update operator

The missing piece of the proof is to check that (2.31) and (2.32) fulfill the condition (2.70). We start by verifying the condition (2.70) for (2.32), which we have presented as special case of (2.31) for linear encoding functions: $\boldsymbol{e}\,(\boldsymbol{\nu} + \boldsymbol{\nu'}) = \boldsymbol{e}\,(\boldsymbol{\nu}) + \boldsymbol{e}\,(\boldsymbol{\nu'})$. Using that property, one can in fact derive (2.32) from (2.31) directly. We now apply (2.32) to $|\boldsymbol{e}(\boldsymbol{\nu})\rangle$, but firstly we note that

$$X_j\,|\boldsymbol{\omega}\rangle = |\boldsymbol{\omega} + \boldsymbol{u_j}\rangle\ , \tag{2.71}$$

where $\boldsymbol{u_j}$ is the j-th unit vector of $\mathbb{Z}_2^{\otimes n}$. Using (2.71) and the linearity of $\boldsymbol{e}$, we find:

$$\begin{aligned} \mathcal{U}^{\boldsymbol{a}}\,|\boldsymbol{e}\,(\boldsymbol{\nu})\rangle &= \left[\bigotimes_{i=1}^{n} (X_i)^{\sum_j A_{ij} q_j}\right] |\boldsymbol{e}(\boldsymbol{\nu})\rangle & (2.72)\\ &= \left[\bigotimes_{i=1}^{n} (X_i)^{\boldsymbol{e}(\boldsymbol{q})}\right] |\boldsymbol{e}(\boldsymbol{\nu})\rangle & (2.73)\\ &= |\boldsymbol{e}(\boldsymbol{\nu}) + \boldsymbol{e}(\boldsymbol{q})\rangle & (2.74)\\ &= |\boldsymbol{e}(\boldsymbol{\nu} + \boldsymbol{q})\rangle\ , & (2.75) \end{aligned}$$

which shows (2.70) for linear encodings.

We now turn our attention to general encodings and prove the same expression for update operators as defined in (2.31):

$$\begin{aligned}
&\mathcal{U}^{\boldsymbol{a}} \left|\boldsymbol{e}\left(\boldsymbol{\nu}\right)\right\rangle \\
&= \left(\sum_{\boldsymbol{t} \in \mathbb{Z}_2^{\otimes n}} \left[\bigotimes_{i=1}^{n} \left(X_i\right)^{t_i} \right] \prod_{j=1}^{n} \frac{1}{2} \left(\mathbb{I} + (-1)^{t_j} \, \mathfrak{X} \left[\varepsilon_j^{\boldsymbol{q}} \right] \right) \right) \left|\boldsymbol{e}\left(\boldsymbol{\nu}\right)\right\rangle && (2.76) \\
&= \left(\sum_{\boldsymbol{t} \in \mathbb{Z}_2^{\otimes n}} \left[\bigotimes_{i=1}^{n} \left(X_i\right)^{t_i} \right] \prod_{j=1}^{n} \underbrace{\frac{1}{2} \left(1 + (-1)^{t_j + \varepsilon_j^{\boldsymbol{q}}(\boldsymbol{e}(\boldsymbol{\nu}))} \right)}_{\delta_{t_j \varepsilon_j^{\boldsymbol{q}}(\boldsymbol{e}(\boldsymbol{\nu}))}} \right) \left|\boldsymbol{e}\left(\boldsymbol{\nu}\right)\right\rangle && (2.77) \\
&= \left(\bigotimes_{i=1}^{n} \left(X_i\right)^{\varepsilon_i^{\boldsymbol{q}}(\boldsymbol{e}(\boldsymbol{\nu}))} \right) \left|\boldsymbol{e}\left(\boldsymbol{\nu}\right)\right\rangle && (2.78) \\
&= \left|\boldsymbol{e}\left(\boldsymbol{\nu}\right) + \boldsymbol{\varepsilon}^{\boldsymbol{q}}\left(\boldsymbol{e}\left(\boldsymbol{\nu}\right)\right)\right\rangle && (2.79) \\
&= \left|\boldsymbol{e}\left(\boldsymbol{\nu}\right) + \boldsymbol{e}\left(\boldsymbol{\nu}\right) + \boldsymbol{e}\left(\boldsymbol{d}\left(\boldsymbol{e}\left(\boldsymbol{\nu}\right)\right) + \boldsymbol{q}\right)\right\rangle && (2.80) \\
&= \left|\boldsymbol{e}\left(\boldsymbol{\nu} + \boldsymbol{q}\right)\right\rangle \,, && (2.81)
\end{aligned}$$

which completes the proof. We swiftly recap what has happened: in (2.76), we have plugged the definition of(2.31) into the left-hand side of (2.70). In between this equation and (2.77), we have evaluated the expectation values of the extracted operators $\mathfrak{X}[\varepsilon_j^{\boldsymbol{q}}]$. From that line to the next, the $\mathbb{Z}_2^{\otimes n}$-sum is collapsed over the condition $\boldsymbol{t} = \boldsymbol{\varepsilon}^{\boldsymbol{q}}(\boldsymbol{e}(\boldsymbol{\nu}))$. We go from (2.78) to (2.79) by applying (2.71). Once we insert the definition (2.29) into (2.79), it becomes obvious that the condition (2.70) is fulfilled. Thus, the entire operator transform is now proven.

2.7.2 Transforming particle-number conserving Hamiltonians

In this section, we examine the richest symmetry to exploit for qubit savings: particle conservation. We begin by introducing the most relevant class of Hamiltonians that exhibit this symmetry, but ultimately the main goal of this section is to simplify the operator transform for all such Hamiltonians. Motivated by the compartmentalized recipes of the conventional mappings, (2.12), we suggest alternatives to the transform (2.30), that do not depend on the sequence length l.

Let us start by noting how easy it is to state that a Hamiltonian the total number of particles: a Hamiltonian like (2.10), conserves the total number of particles when every term $\widehat{h}_{\boldsymbol{ab}}$ has as many creation operators as it has annihilation operators. The lengths l, implicit in the sequences $\widehat{h}_{\boldsymbol{ab}}$ that occur in the Hamiltonian, are thereby determined by the field theory or model, that underlies the problem. The coefficients $h_{\boldsymbol{ab}}$, on the other hand, are determined by the set of basis functions used. For first-principle problems in quantum chemistry and solid state physics, we usually encounter particle-number-conserving Hamiltonians with terms of weight that is at most $l = 4$ (1.8). In the notation of (2.10), these coefficients V_{ijkl} and t_{ij} correspond to $h_{(i,j,k,l)(1,1,0,0)}$ and $h_{(i,j)(1,0)}$. The $(l = 4)$ interaction terms usually originate from either magnetism and/or the Coulomb interaction. Even for these $(l = 4)$-terms, the operator transform (2.30) is quite bulky, and we in general would like to have a transform that is independent of l. Before we begin to discuss such transform recipes however, we need to set up some preliminaries. First of all, we need to find a suitable code $(\boldsymbol{e}, \boldsymbol{d})$, as discussed in the main part. Ideally, we would encode only the Hilbert space with the correct number of particles, M, but Hilbert spaces of other particle numbers can also be included. Assuming that the Hamiltonian visits every state with the same particle number, we must encode entire Hilbert spaces $\mathcal{H}_N^m$ only. Secondly, we need to reorder the fermionic operators inside the Hamiltonian terms $\widehat{h}_{\boldsymbol{ab}}$. The reason for this is, that our goal can only be achieved by finding recipes for smaller sequences of constant length. In order to transform the Hamiltonian terms then, we need to invoke the anticommutation relations (1.7) to introduce an order in $\widehat{h}_{\boldsymbol{ab}}$, such that these small sequences appear as consecutive, distinct blocks. As we shall see, these blocks will have the shape $c_i^\dagger c_j$. So every $\widehat{h}_{\boldsymbol{ab}}$ needs to be reordered, such that every even operator is a creation operator, and every odd operator an annihilator. For the $(l = 4)$-terms in (1.8), this reordering means $c_i^\dagger c_j^\dagger c_k c_l \;\mapsto\; c_i^\dagger c_l\, c_j^\dagger c_k - \delta_{jl}\; c_i^\dagger c_k$.

Let us quickly sketch the idea behind that reordering and introduce some nomenclature: instead of considering Hamiltonian terms, we realize that also the terms $c_i^\dagger c_j$ also conserve the particle number: $\mathcal{H}_N^m \mapsto \mathcal{H}_N^m$. Let us act with $c_i^\dagger c_j$ on an encoded state. We consider a state that is not annihilated by $c_i^\dagger c_j$. Its particle number is reduced by one through c_j, but then immediately restored by $c_i^\dagger$. In fact, for a general sequence of that arrangement, every even operator restores the particle number in this way and every odd reduces it. We therefore call the subspace, in which we find the state after an even (odd) number of operators, the even (odd) subspace. Since all l must be even for the Hamiltonian to have particle conservation symmetry, the even subspace is the one encoded. The odd subspace, on the other hand, has one particle less, so it is $\mathcal{H}_N^{(M-1)}$, if the even one is $\mathcal{H}_N^M$.

2.7.2.1 Encoding the two spaces separately

In this ordering, one can find a recipe for a singular creation or annihilation operator. The strategy is to consider a second code for the odd subspace. As before $(\boldsymbol{e}, \boldsymbol{d})$ denotes the code for the even subspace, and now $(\boldsymbol{e'}, \boldsymbol{d'})$ is encoding the odd subspace. The idea is that after an odd operator (which in this ordering is an annihilation operator), the state is updated into the odd subspace. With every even operator (which is a creation operator), the state is updated from the odd subspace back into the even one. We find:

$$\begin{aligned} c_j^\dagger \quad &\hat{=} \quad \frac{1}{2}\,\bar{\mathcal{U}}^{(j)}\ \left(\mathbb{I} + \mathfrak{X}\left[d_j\right]\right)\ \mathfrak{X}\left[p_j\right]\,, && (2.82) \\ c_j \quad &\hat{=} \quad \frac{1}{2}\,\mathcal{U}^{(j)}\ \left(\mathbb{I} - \mathfrak{X}\left[d_j'\right]\right)\mathfrak{X}\left[p_j'\right]\,. && (2.83) \end{aligned}$$

In (2.83), $\mathcal{U}^{(j)}$ is defined as in (2.31), but its counterpart from (2.82) is defined by

$$\bar{\mathcal{U}}^{(j)} = \sum_{\boldsymbol{t}\,\in\,\mathbb{Z}_2^{\otimes n}} \left[\bigotimes_{i=1}^{n} (X_i)^{t_i}\right] \prod_{i=1}^{n} \frac{1}{2}\left(\mathbb{I} + (-1)^{t_i}\ \mathfrak{X}\left[\varepsilon_k'^{\,\boldsymbol{u_j}}\right]\right)\,, \qquad (2.84)$$

with the primed functions $\boldsymbol{\varepsilon}'^{\,\boldsymbol{q}}$, $\boldsymbol{p}'$ defined like (2.29) and (2.28), but with $(\boldsymbol{e'}, \boldsymbol{d'})$ in place of $(\boldsymbol{e}, \boldsymbol{d})$.

This method relies on n qubits being feasible to simulate the odd subspace in. That is, however, not always the case. The basis set of $\mathcal{H}_N^{M-1}$ is in general larger than $\mathcal{H}_N^M$, when $M > N/2$. In this way, the odd subspace can also be larger and even be infeasible to simulate with just n qubits. As a solution, one changes the ordering into odd operators being creation operators, and even ones being annihilators, like $c_k c_i^\dagger\, c_l\, c_j^\dagger$. This causes the odd subspace to become $\mathcal{H}_N^{(M+1)}$, which has a smaller basis set than $\mathcal{H}_N^M$. For that case $(\boldsymbol{e}, \boldsymbol{d})$ become the code for the odd subspace, and $(\boldsymbol{e'}, \boldsymbol{d'})$ will be associated to the even subspace in (2.82) and (2.83).

The obvious disadvantage is that two codes have to be employed at once.

However, the checksum code for instance (Section2.4.3.1 in the main part), comes in two different flavors already, which can be used as codes for even and odd subspaces, respectively.

2.7.2.2 Encoding the building blocks

The building blocks $c_i^\dagger c_j$ are guaranteed to conserve the particle number, so the even subspace is conserved. As a consequence, one may consider the possibility to transform the operators as the pairs we have rearranged them into. In this way, we still have a certain compartmentalization of (2.30). Two special cases are to be taken into account: when $i > j$, an additional minus sign has to be added, as compared to the $i < j$ case. Also, when $i = j$, all parity operators cancel and the projectors coincide. We find:

$$c_i^\dagger c_j \mathrel{\hat{=}} \begin{cases} \frac{1}{4}\,(-1)^{\theta_{ij}}\,\mathcal{U}^{(i,j)}\,\mathfrak{X}\left[p_i + p_j\right]\,\left(\mathbb{I} + \mathfrak{X}\left[d_i\right]\right)\left(\mathbb{I} - \mathfrak{X}\left[d_j\right]\right) & i \neq j \\ \\ \frac{1}{2}\left(1 - \mathfrak{X}\left[d_j\right]\right) & i = j\,, \end{cases} \tag{2.85}$$

with $\mathcal{U}^{(i,j)}$ being the $l = 2$ version of (2.31), and $\boldsymbol{p}$ and $\boldsymbol{\varepsilon}^{\boldsymbol{q}}$ defined as usual by (2.28) and (2.29).

2.7.3 Multi-weight binary addressing codes based on dissections

With binary addressing codes, that is codes that are similar to the one presented in Section 2.4.3.2 in the main part, even an exponential amount of qubits can be saved for systems with low particle number, but at the expense of complicated gates. For this section, we firstly recap the situation of Section 2.4.3.2 and clarify what binary addressing means. Firstly, some nomenclature is introduced. We then generalize the concept of binary addressing codes to weight-K codes, using results from [31]. As an example, we explicitly obtain the $K = 2$ code.

Suppose we have a system with $N = 2^r$ orbitals, and one particle in it. Our goal is to encode the basis state, where the particle is on orbital $y \in [2^r]$, as a binary number in r qubits. In this way, the state with occupational vector $\boldsymbol{u_y}$ is encoded as $|\boldsymbol{q}^{\boldsymbol{y},\boldsymbol{r}}\rangle$, with $\boldsymbol{q}^{\boldsymbol{y},\boldsymbol{r}} \in \mathbb{Z}_2^{\otimes r}$ and

$y = \mathrm{bin}(\boldsymbol{q}^{\boldsymbol{y},\boldsymbol{r}}) + 1$. Probing an unknown basis state, a decoding will now have components of the form

$$\boldsymbol{\omega} \mapsto \prod_{i \in [r]} (\omega_i + q_i^{y,r} + 1) \,. \tag{2.86}$$

Such binary functions output 1 only when $\boldsymbol{\omega} = \boldsymbol{q}^{\boldsymbol{y},\boldsymbol{r}}$. In our nomenclature, we say that in the basis state $|\boldsymbol{q}^{\boldsymbol{y},\boldsymbol{r}}\rangle$, the particle has the coordinate y. We refer to codes that store particle coordinates in binary form, as binary addressing codes.

In the $K = 1$ case from the main part, the code words just contain the binary representation of one coordinate. The question is now how to generalize the binary addressing codes. For multi-weight codes, we have to have K sub-registers to store the addresses of K particles. Naively, one would want to store the coordinate of each particle in its respective sub-register in binary form, as we have done for $K = 1$. This however, holds a problem. As the particles are indistinguishable, the stored coordinates would be interchangeable, the code would not be one-to-one. For the binary numbers $\boldsymbol{\omega}^1$ and $\boldsymbol{\omega}^2$, that represent a coordinate each, this would mean $\boldsymbol{d}(\boldsymbol{\omega}^1 \oplus \boldsymbol{\omega}^2) = \boldsymbol{d}(\boldsymbol{\omega}^2 \oplus \boldsymbol{\omega}^1)$. That strategy not only complicates the operator transform, it also leads to a certain qubit overhead, as each plain word has as many code words as there are permutations of K items. Since this naïve idea leaves us unconvinced, we abandon it and search for one-to-one codes instead. The key is to consider the coordinates to be in a certain format and this is where [31] comes into play. We proceed by using some relevant concepts of that paper.

Let us consider the coordinates of K particles to be given in the N-ary vector $\boldsymbol{x} = (x_1, \ldots, x_K)$. Between those coordinates, we have imposed an ordering $x_i > x_j$ as $i > j$. Particles cannot share the same orbital, so we are excluding the cases where two coordinates are equal. Using results from [31], we transform the latter into coordinates that lack such an ordering, and where each component is an integer from a different range:

$$\boldsymbol{x} \;\mapsto\; \boldsymbol{y} = (y_1, \ldots, y_K)^\top \qquad \text{with} \quad \boldsymbol{y} \in \bigotimes_{m=1}^{K} \left[\left[\left\lceil \frac{N}{m} \right\rceil \right]\right] \,. \tag{2.87}$$

Through that transform, each vector $\boldsymbol{y}$ corresponds to a valid vector $\boldsymbol{x}$, and there is no duplication. We now represent the $\boldsymbol{y}$-coordinates by binary numbers in the code words $\boldsymbol{\omega} \in \mathbb{Z}_2^{\otimes n}$, where $n = \sum_{m=1}^{K} \lceil \log \frac{N}{m} \rceil$:

$$\boldsymbol{\omega} = \bigoplus_{m=1}^{K} \boldsymbol{q}^{\,\boldsymbol{y}_m, \lceil \frac{N}{m} \rceil} \qquad \text{with} \quad \boldsymbol{q}^{\boldsymbol{i},\boldsymbol{j}} \in \mathbb{Z}_2^{\otimes j} \quad \text{and} \quad \text{bin}\left(\boldsymbol{q}^{\boldsymbol{i},\boldsymbol{j}}\right) + 1 = i \,. \tag{2.88}$$

A geometric interpretation of the process portrays the vector $\boldsymbol{x}$ as a set of coordinates in a K-dimensional, discrete vector space. The vectors allowed by the ordering form thereby a multi-dimensional tetrahedron. The states outside the tetrahedron do not correspond to a valid $\mathcal{V}$ vector, so encoding each coordinate x_i in $\lceil \log N \rceil$ qubits would be redundant. We therefore dissect the tetrahedron, and rearrange it into a *brick*, as it is referred to in [31]. What is actually done is to apply symmetry operations (like point-reflections) on the vector space until the tetrahedron is deformed into the desired shape a K-dimensional, rectangular volume. The fact that the vectors to encode are now all inside a hyper-rectangle is what we wanted to achieve. We can now clip the ranges of the coordinate axes (to $[\lceil \log \frac{N}{m} \rceil]$) to exclude vectors the vectors outside the brick. As the values on the axes correspond to non-binary addresses, this means that the qubit space is trimmed as well, and we have eliminated all states based on not-allowed coordinates. This is where we now reconnect to our task of finding a code: the $\boldsymbol{e}$- and $\boldsymbol{d}$-functions have to take into account the reshaping process, as only the coordinates $\boldsymbol{x}$ have a physical interpretation and can be decoded. The binary addresses in the code words, on the other hand, are representatives of $\boldsymbol{y}$. With binary logic, the two coordinates have to be reconnected. We illustrate this abstract process on the example of the $(K = 2)$-code.

Weight-two binary addressing code

As an example, we present the weight-two binary addressing code on $N = 2^r$ orbitals. The integer r will determine the size of the entire qubit system $n = 2r - 1$, with two registers of size r and $r - 1$.

With the two registers, a binary vector $\boldsymbol{\omega} = \boldsymbol{\alpha} \oplus \boldsymbol{\beta}$ with $\boldsymbol{\alpha} \in \mathbb{Z}_2^{\otimes r}$ and $\boldsymbol{\beta} \in \mathbb{Z}_2^{\otimes (r-1)}$ is defining the qubit basis. In two dimensions, the brick turns into a rectangle and the tetrahedron into triangle. The decoding function takes binary addresses of the rectangular $\boldsymbol{y}$, and transforms them into coordinates in the triangle $\boldsymbol{x}$. The ordering condition implies hereby where to dissect the rectangle: Figure 2.2 may serve as a visual aid, disregard-

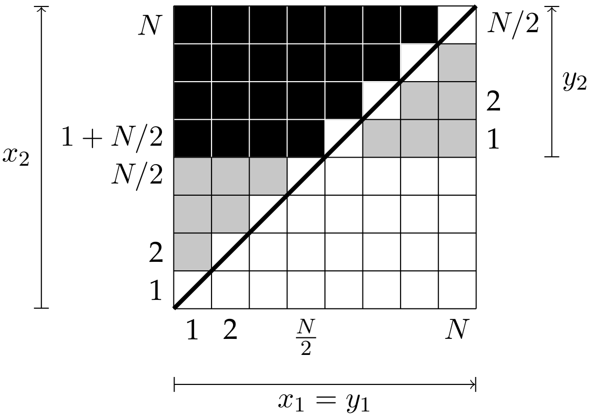


Figure 2.2. Visualization of the 2-dimensional vector space: a valid vector is represented as a colored tile. The left gray tiles and the black ones constitute the triangle, defining all valid vectors $\boldsymbol{x} = (x_1, x_2)^\top$. The marked diagonal tiles are to be excluded from the encoded space. The black tiles and the gray ones on the right of this diagonal form the brick, containing all $\boldsymbol{y} = (y_1, y_2)^\top$ vectors.

ing the excluded cases of $y_1 = y_2$, we find for $y_1 \in [N]$, $y_2 \in [N/2]$ and $\boldsymbol{x} \in [N]^{\otimes 2}$:

$$(x_1, x_2) = \begin{cases} (y_1, N/2 + y_2) & \text{for} \quad y_1 < N/2 + y_2 \\ (y_1, N/2 - y_2 + 1) & \text{for} \quad y_1 > N/2 + y_2 \,. \end{cases} \tag{2.89}$$

This decoding is translated into a binary functions as follows: the coordinate y_1 is represented by the binary vector $\boldsymbol{\alpha}$ and y_2 by $\boldsymbol{\beta}$. For each component defined by the binary vector $\boldsymbol{b} \in \mathbb{Z}_2^{\otimes r}$, we have

$$\begin{aligned} d_j\left(\boldsymbol{\alpha} \oplus \boldsymbol{\beta}\right) \;=\; & S\left(\boldsymbol{\alpha}, \boldsymbol{\beta}\right) \prod_{i=1}^{r} \left(\alpha_i + q_i^{j,r} + 1\right) \\ & + \left(1 + S\left(\boldsymbol{\alpha}, \boldsymbol{\beta}\right)\right)\left(1 + T\left(\boldsymbol{\alpha}, \boldsymbol{\beta}\right)\right) \prod_{i=1}^{r} \left(\alpha_i + q_i^{j,r}\right) \\ & + \left(1 + S\left(\boldsymbol{\alpha}, \boldsymbol{\beta}\right)\right)\left(1 + T\left(\boldsymbol{\alpha}, \boldsymbol{\beta}\right)\right) \prod_{k=1}^{r-1} \left(\beta_k + q_k^{j,r}\right) \\ & + S\left(\boldsymbol{\alpha}, \boldsymbol{\beta}\right) \prod_{k=1}^{r-1} \left(\beta_k + q_k^{j,r} + 1\right) \,, \end{aligned} \tag{2.90}$$

with $\boldsymbol{q}^{\boldsymbol{j,r}} = (q_1^{j,r}, q_2^{j,r}, \dots, q_{2^r}^{j,r})$ as defined in (2.88) and we have employed two binary functions S and $T : (\mathbb{Z}_2^{\otimes r}, \mathbb{Z}_2^{\otimes (r-1)}) \mapsto \mathbb{Z}_2$. Here, S compares the binary numbers to determine if the coordinates are left of the dissection (a black tile in Figure 2.2).

$$S\left(\boldsymbol{\alpha}, \boldsymbol{\beta}\right) = \alpha_r \sum_{j=1}^{r-1} \left[\prod_{r-1 \geq i > j} (\alpha_i + \beta_i + 1) \right] (1 + \alpha_j)\beta_j + 1 + \alpha_r \tag{2.91}$$

The binary function T, on the other hand, is checking whether a set of coordinates is on a diagonal position (diagonally marked tiles). These excluded cases are mapped to $(0)^{\otimes r}$ altogether.

$$T\left(\boldsymbol{\alpha}, \boldsymbol{\beta}\right) = \prod_i (\alpha_i + \beta_i) \tag{2.92}$$

This concludes the decoding function. Unfortunately, the amount of logic elements in the decoding will complicate the weight-two codes quite a bit, and the encoding function is hardly better. The reason for this is to find in the ordering condition: the update operations are conditional on whether we change the ordering of the coordinates represented by $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. This is reflected in a nonlinear encoding function: we remind us that the encoding function is a map $\boldsymbol{e} : \mathbb{Z}_2^{\otimes 2^r} \mapsto \mathbb{Z}_2^{\otimes (2r-1)}$, and with $\boldsymbol{\nu} \in \mathbb{Z}_2^{\otimes 2^r}$ we find

$$\begin{aligned} \boldsymbol{e}\left(\boldsymbol{\nu}\right) \quad = \quad & \sum_{j=2}^{2^{r-1}} \sum_{i=1}^{j-1} \left(\boldsymbol{q}^{\boldsymbol{i,r}} + \boldsymbol{I}^{\boldsymbol{r}} \right) \oplus \left(\boldsymbol{q}^{\boldsymbol{j,r-1}} + \boldsymbol{I}^{\boldsymbol{r-1}} \right) \nu_i \nu_j \\ & + \sum_{j=2^{r-1}+1}^{2^r} \sum_{i=1}^{2^{r-1}} \left(\boldsymbol{q}^{\boldsymbol{i,r}} + \boldsymbol{I}^{\boldsymbol{r}} \right) \oplus \left(\boldsymbol{q}^{\boldsymbol{j-2^{r-1},r-1}} \right) \nu_i \nu_j \\ & + \sum_{j=2^{r-1}+2}^{2^r} \sum_{i=2^{r-1}+1}^{j-1} \left(\boldsymbol{q}^{\boldsymbol{i,r}} \right) \oplus \left(\boldsymbol{q}^{\boldsymbol{j-2^{r-1},r-1}} \right) \nu_i \nu_j \,, \end{aligned}$$

with $\boldsymbol{q}^{\boldsymbol{i,j}}$ as defined in (2.88), and $\boldsymbol{I}^{\boldsymbol{j}} = (1)^{\otimes j} = \boldsymbol{q}^{\boldsymbol{2^j,j}}$.

The dissecting of tetrahedrons can be generalized for codes of weight larger than two (see again [31]), but as one increases the number of dissections, the code functions are complicated even further.

2.7.4 Segment codes

In this section, we provide detailed information on the segment codes. We firstly concern ourselves with the segmentation of the global code, including a derivation of the segment sizes. In another subsection we construct the segment codes themselves. The last subsection is dedicated to the adjustments one has to make to Hamiltonian, such that segment codes become feasible to use.

2.7.4.1 Segment sizes

At this point we want to sketch the idea behind the segment sizes ($\widehat{N}$, $\widehat{n}$) stated during Section 2.4.3.3 in the main part, but first of all we would like to clearly set up the situation.

We consider vectors $\boldsymbol{\nu} \in \mathbb{Z}_2^{\otimes N}$ to consist of $\widehat{m}$ smaller vectors $\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$ of length $\widehat{n}+1$, such that $\boldsymbol{\nu} = \bigoplus_{i=1}^{\widehat{m}} \widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$. We call those vectors $\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$ segments of $\boldsymbol{\nu}$. The goal is now to find a code ($\boldsymbol{e}$, $\boldsymbol{d}$) to encode a basis $\mathcal{V}$ which contains all vectors $\boldsymbol{\nu}$ with Hamming weight K. For that purpose we relate the segment $\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$ to a segment of the code space, $\widehat{\boldsymbol{\omega}}^{\boldsymbol{i}}$, for all $i \in [N]$. The code space segments constitute the code words in a fashion similar to the previous segmentation of $\boldsymbol{\nu}$: $\boldsymbol{\omega} = \bigoplus_{i=1}^{\widehat{m}} \widehat{\boldsymbol{\omega}}^{\boldsymbol{i}}$. However, the length of those binary vectors $\widehat{\boldsymbol{\omega}}^{\boldsymbol{i}}$ is $\widehat{n}$, such that with $n = \widehat{m}\widehat{n}$ and $N = \widehat{m}(\widehat{n}+1)$, the problem is reduced by $\widehat{m}$ qubits as compared to conventional transforms. We now introduce the *subcodes* ($\widehat{\boldsymbol{e}} : \mathbb{Z}_2^{\otimes(\widehat{n}+1)} \mapsto \mathbb{Z}_2^{\otimes\widehat{n}}, \widehat{\boldsymbol{d}} : \mathbb{Z}_2^{\otimes\widehat{n}} \mapsto \mathbb{Z}_2^{\otimes(\widehat{n}+1)}$), with which we encode the i-th segment $\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$ as $\widehat{\boldsymbol{\omega}}^{\boldsymbol{i}}$ (see Figure 2.3). Note that we require the subcodes to inherit all the code properties. In this way we guarantee the code properties of the *global code* ($\boldsymbol{e}$, $\boldsymbol{d}$) when appending $\widehat{m}$ instances of the same subcode:

$$\boldsymbol{d}\left(\bigoplus_{i=1}^{\widehat{m}} \widehat{\boldsymbol{\omega}}^{\boldsymbol{i}}\right) = \bigoplus_{i=1}^{\widehat{m}} \widehat{\boldsymbol{d}}\left(\widehat{\boldsymbol{\omega}}^{\boldsymbol{i}}\right) , \qquad \boldsymbol{e}\left(\bigoplus_{i=1}^{\widehat{m}} \widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}\right) = \bigoplus_{i=1}^{\widehat{m}} \widehat{\boldsymbol{e}}\left(\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}\right) . \tag{2.93}$$

The orbital number being an integer multiple of the block size is of course an idealized scenario. One will probably have to add a few other components in order to compensate for dimensional mismatches.

We now set out to find the smallest segment size $\widehat{n}$. It should be clear that $\widehat{n}$ is a function of the targeted Hamming weight K: this means K determines which segment codes are suitable for the system. The reason for

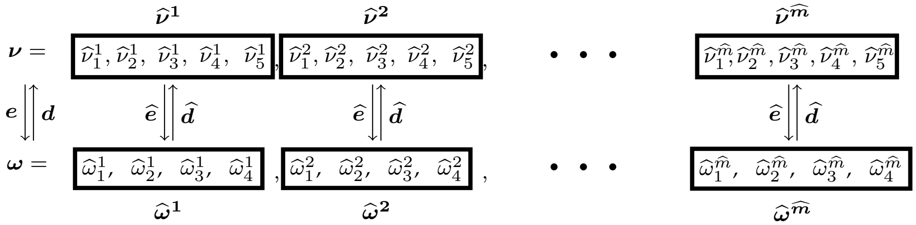

$\widehat{\boldsymbol{\nu}}^{\boldsymbol{1}}$
$\widehat{\boldsymbol{\nu}}^{\boldsymbol{2}}$
$\widehat{\boldsymbol{\nu}}^{\boldsymbol{\widehat{m}}}$
$\boldsymbol{\nu} =$
$\widehat{\nu}_1^1, \widehat{\nu}_2^1, \widehat{\nu}_3^1, \widehat{\nu}_4^1, \widehat{\nu}_5^1$
$\widehat{\nu}_1^2, \widehat{\nu}_2^2, \widehat{\nu}_3^2, \widehat{\nu}_4^2, \widehat{\nu}_5^2$
$\widehat{\nu}_1^{\widehat{m}}, \widehat{\nu}_2^{\widehat{m}}, \widehat{\nu}_3^{\widehat{m}}, \widehat{\nu}_4^{\widehat{m}}, \widehat{\nu}_5^{\widehat{m}}$
$\boldsymbol{e}$
$\boldsymbol{d}$
$\widehat{\boldsymbol{e}}$
$\widehat{\boldsymbol{d}}$
$\boldsymbol{\omega} =$
$\widehat{\omega}_1^1, \widehat{\omega}_2^1, \widehat{\omega}_3^1, \widehat{\omega}_4^1$
$\widehat{\omega}_1^2, \widehat{\omega}_2^2, \widehat{\omega}_3^2, \widehat{\omega}_4^2$
$\widehat{\omega}_1^{\widehat{m}}, \widehat{\omega}_2^{\widehat{m}}, \widehat{\omega}_3^{\widehat{m}}, \widehat{\omega}_4^{\widehat{m}}$
$\widehat{\boldsymbol{\omega}}^{\boldsymbol{1}}$
$\widehat{\boldsymbol{\omega}}^{\boldsymbol{2}}$
$\widehat{\boldsymbol{\omega}}^{\boldsymbol{\widehat{m}}}$

Figure 2.3. Visualization of (2.93) for $\widehat{n} = 4$. The global code $(\boldsymbol{e}, \boldsymbol{d})$ relates the occupation vectors to the global code words $\boldsymbol{\nu} \leftrightarrow \boldsymbol{\omega}$. The an instance of the subcode $(\widehat{\boldsymbol{e}}, \widehat{\boldsymbol{d}})$ relates i-th block in $\boldsymbol{\nu}$, $\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$, to the i-th segment in the code words, $\widehat{\boldsymbol{\omega}}^{\boldsymbol{i}}$.

this is that we need to encode all vectors with weight 0 to K inside every segment, taking into account for the up to K particles on the orbitals inside one segment. In order to include weight-K vectors, the size of each segment must be at least K. If the segment size would be exactly K, on the other hand, we end up encoding the entire Fock space again. In doing so, we are not making any qubit savings. The segments must thus be larger than K. In other words, we look for an integer $\widehat{n} > K$, where the sum of all combinations $\widehat{\boldsymbol{\nu}} \in \mathbb{Z}_2^{\otimes(\widehat{n}+1)}$ with $\mathrm{w_H}\left(\widehat{\boldsymbol{\nu}}\right) \leq K$ is smaller equal $2^{\widehat{n}}$.

$$2^{\widehat{n}} \geq \sum_{k=0}^{K} \binom{\widehat{n}+1}{k} \tag{2.94}$$

In the case $\widehat{n} = 2K$, the condition is fulfilled as identity, since exactly half of all $2^{\widehat{n}+1}$ combinations are included in the sum.

2.7.4.2 Subcodes

This subsection offers a closer look at the construction of the segment subcodes $(\widehat{\boldsymbol{e}}, \widehat{\boldsymbol{d}})$. Let us start by considering the decoding $\widehat{\boldsymbol{d}}$ in order to explore the nature of the binary switch $f(\widehat{\boldsymbol{\omega}})$, that occurs in (2.37). One observes the two (affine) linear $\left(\mathbb{Z}_2^{\otimes \widehat{n}} \mapsto \mathbb{Z}_2^{\otimes (\widehat{n}+1)}\right)$-maps

$$\widehat{\boldsymbol{\omega}} \mapsto \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \\ 0 & \dots & 0 \end{bmatrix} \widehat{\boldsymbol{\omega}} \,, \qquad \widehat{\boldsymbol{\omega}} \mapsto \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \\ 0 & \dots & 0 \end{bmatrix} \widehat{\boldsymbol{\omega}} + \begin{pmatrix} 1 \\ \vdots \\ \vdots \\ 1 \end{pmatrix} . \tag{2.95}$$

to produce together all the vectors with weight equal or smaller than K, if we input all $\widehat{\boldsymbol{\omega}}$ with $\mathrm{w_H}\left(\widehat{\boldsymbol{\omega}}\right) \leq K$ into the first, and the remaining cases with $\mathrm{w_H}\left(\widehat{\boldsymbol{\omega}}\right) > K$ into the second one. Note that the last component is always zero in outputs of the first function and one in the second. Therefore, the inverse of both maps is always a linear map with the matrix $\left[\, \mathbb{I} \mid \boldsymbol{I}^{\widehat{n}} \right]$. We take this inverse as encoding (2.38), and the two maps (2.95) are merged into the decoding (2.37). In order to switch between these two maps we define the binary function $f(\widehat{\boldsymbol{\omega}}) : \mathbb{Z}_2^{\otimes \widehat{n}} \mapsto \mathbb{Z}_2$ such that

$$f(\widehat{\boldsymbol{\omega}}) = \begin{cases} 1 & \text{for } \ \mathrm{w_H}\left(\widehat{\boldsymbol{\omega}}\right) > K \\ 0 & \text{otherwise} \,. \end{cases} \tag{2.96}$$

In general, one can define this binary switch in a brute-force way by

$$f\left(\widehat{\boldsymbol{\omega}}\right) = \sum_{k=K+1}^{2K} \sum_{\substack{\boldsymbol{t} \in \mathbb{Z}_2^{\otimes 2K} \\ \mathrm{w_H}(\boldsymbol{t})=k}} \prod_{m=1}^{2K} \left(\widehat{\omega}_m + 1 + t_m\right) \,. \tag{2.97}$$

For the case $K = 1$ ($\widehat{n} = 2$), the switch equals $f(\boldsymbol{\omega}) = \omega_1 \omega_2$, and for the code we recover a version of binary addressing codes, where the vector $(0, 0, 0)$ is encoded.

$$\widehat{\boldsymbol{d}}\left(\widehat{\boldsymbol{\omega}}\right) = \begin{pmatrix} \widehat{\omega}_1 \left(\widehat{\omega}_2 + 1\right) \\ \left(\widehat{\omega}_1 + 1\right) \widehat{\omega}_2 \\ \widehat{\omega}_1 \widehat{\omega}_2 \end{pmatrix} , \qquad \widehat{\boldsymbol{e}}\left(\widehat{\boldsymbol{\nu}}\right) = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \widehat{\boldsymbol{\nu}} \,. \tag{2.98}$$

In the $K = 2$ ($\widehat{n} = 4$) case, this binary switch is found to be $f\left(\widehat{\boldsymbol{\omega}}\right) = \widehat{\omega}_1 \widehat{\omega}_2 \widehat{\omega}_3 + \widehat{\omega}_1 \widehat{\omega}_2 \widehat{\omega}_4 + \widehat{\omega}_1 \widehat{\omega}_3 \widehat{\omega}_4 + \widehat{\omega}_2 \widehat{\omega}_3 \widehat{\omega}_4 + \widehat{\omega}_1 \widehat{\omega}_2 \widehat{\omega}_3 \widehat{\omega}_4$.

2.7.4.3 Hamiltonian adjustments

As mentioned in Section 2.4.3.3, in the main part, segment codes are not automatically compatible with all particle-number-conserving Hamiltonians. We show here, how certain adjustments can be made to these

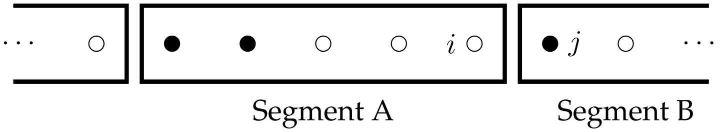


Figure 2.4. (Filled) Circles represent (occupied) fermionic orbitals, where $K = 2$ segment codes are used in the indicated blocks. This occupational case is problematic for the codes, as the operator $c_i^\dagger c_j$ acting on this state leaves the encoded space.

Hamiltonians, such that their action on the space $\mathcal{H}_N^K$ is not changed, but segment codes become feasible to describe them with. In order to understand this issue, we begin by examining the encoded space. For that purpose we reprise the situation of (2.93), where we have appended $\widehat{m}$ instances of the same subcode. With segment codes, the basis $\mathcal{V}$ contains vectors with Hamming weights from 0 to $\widehat{m}K$. We have encoded all possible vectors $\boldsymbol{\nu}$ with $0 \leq \mathrm{w_H}\left(\boldsymbol{\nu}\right) \leq K$, but although we have some, not all vectors with $\mathrm{w_H}\left(\boldsymbol{\nu}\right) > K$ are encoded. We can illustrate that point rather quickly: each segment has length $2K+1$, but the subcode encodes vectors $\widehat{\boldsymbol{\nu}}$ with only $\mathrm{w_H}\left(\widehat{\boldsymbol{\nu}}\right) \leq K$. The (global) basis $\mathcal{V}$ is thus deprived of vectors $\boldsymbol{\nu} = (\bigoplus_i \widehat{\boldsymbol{\nu}}^{\boldsymbol{i}})$ where for any segment i, $\mathrm{w_H}\left(\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}\right) > K$.

We now turn our attention to terms, which, when present in a Hamiltonian, make segment codes infeasible to use. Note, that $\mathcal{V}$-vectors with $\mathrm{w_H}\left(\boldsymbol{\nu}\right) \neq K$, are not corresponding to fermionic states we are interested in. In particular it is a certain subset of states with $\mathrm{w_H}\left(\boldsymbol{\nu}\right) > K$, which can lead out of the encoded space (into the states previously mentioned) when acted upon with certain fermionic operators. Let us consider the operator $c_i^\dagger c_j$ as an example, where i and j are in different segments (let us call these segments A and B). Now a basis state as depicted in Figure 2.4, is not annihilated by $c_i^\dagger c_j$, and leads into a state with 3 particles in segment A. The problem is that the initial state is encoded in the $(K = 2)$ segment codes, whereas the updated state (with the 3 particles in A) is not. In general, operators $\widehat{h}_{\boldsymbol{ab}}$, that change occupations in between segments, will cause some basis states with $\mathrm{w_H}\left(\boldsymbol{\nu}\right) > K$ to leave the encoded space. We can however adjust these terms $\widehat{h}_{\boldsymbol{ab}} \to \widehat{h}'_{\boldsymbol{ab}}$, such that $\widehat{h}'_{\boldsymbol{ab}} : \mathrm{span}(\mathcal{B}) \mapsto \mathrm{span}(\mathcal{B})$, where $\mathcal{B}$ is the basis encoded by the segment codes. We now sketch the idea behind those adjustments, before we reconsider the situation of Figure 2.4. Note that after these adjustments have been made to all Hamiltonian terms in question, the segment codes are compatible with the new Hamiltonian. The idea is to switch those terms off for states, that already have K particles inside the segments, to which particles will be added. We have to take care to do this in a way that leaves the Hamiltonian hermitian on the level of second quantization, i.e. we have to adjust the terms $\widehat{h}_{\boldsymbol{ab}}$ and $\widehat{h}^{\dagger}_{\boldsymbol{ab}}$ into $\widehat{h}'_{\boldsymbol{ab}}$ and $(\widehat{h}^{\dagger}_{\boldsymbol{ab}})'$, such that $\widehat{h}'_{\boldsymbol{ab}} + (\widehat{h}^{\dagger}_{\boldsymbol{ab}})'$ is hermitian. For the $K = 2$ code of Figure 2.4, we can make the following adjustments:

$$c_i^\dagger c_j \;\mapsto\; \left(1 - \sum_{l,k<l\,\in\,\mathrm{B}} c_k^\dagger c_k c_l^\dagger c_l\right) c_i^\dagger c_j \left(1 - \sum_{w,v<w\,\in\,\mathrm{A}} c_v^\dagger c_v c_w^\dagger c_w\right) . \tag{2.99}$$

2.8 Notations

[...] The set of integers from 1 to the argument.

$\hat{=}$ Relation used to express the correspondence between fermionic operators/states to qubit counterparts (according to some fermion-to-qubit mapping).

$\boldsymbol{a}$ Element of $[N]^{\otimes l}$. Length-l N-ary vector parametrizing orbitals in the Hamiltonian terms $\widehat{h}_{\boldsymbol{ab}}$.

$A^{(-1)}$ A $(N \times N)$ binary matrix defining a conventional encoding (decoding).

$\boldsymbol{b}$ Element of $\mathbb{Z}_2^{\otimes l}$. A length-l binary vector determining operator types in $\widehat{h}_{\boldsymbol{ab}}$.

$\mathcal{B}$ The basis of a space of fermions on N orbitals, which is possibly smaller than the Fock-space basis.

$c_j^{(\dagger)}$ An element of $\mathcal{L}(\mathcal{F}_N)$. A fermionic annihilation (creation) operator.

$(\mathbb{C}^2)^{\otimes n}$ The vector space of n-qubit states.

$\boldsymbol{d}$ A binary vector function $\mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2^{\otimes N}$. The decoding function.

$\boldsymbol{e}$ Binary vector function $\mathbb{Z}_2^{\otimes N} \mapsto \mathbb{Z}_2^{\otimes n}$. The encoding function.

$\boldsymbol{\varepsilon}^{\boldsymbol{q}}$ A binary vector function $\mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2^{\otimes n}$. The update function which plays a role for nonlinear encodings, see (2.29).

$\mathcal{F}_N$	The fermionic Fock space restricted on N orbitals.
$F(j)$	A Subset of $[n]$ The flip set with respect to orbital j, see (2.12).
$\widehat{h}_{\boldsymbol{ab}}$	A fermion operator $\mathrm{span}(\mathcal{B}) \mapsto \mathrm{span}(\mathcal{B})$. A generic term in a fermionic Hamiltonian, see (2.10).
$\mathcal{H}_N^M$	The antisymmetrized Hilbert space of M indistinguishable fermions on N orbitals.
$\mathbb{I}$	The identity operator on arbitrary spaces.
K	An element of $[N]$ The targeted Hamming weight of the vectors encoded in a binary code.
l	The length of a sequence of fermionic operators in $\widehat{h}_{\boldsymbol{ab}}$.
L	The weight of a Pauli string.
$\mathcal{L}(...)$	Denotes linear operators on the argument vector space.
M	The total number of particles in a system of N orbitals.
n	The number of qubits.
N	The number of fermionic orbitals.
$\boldsymbol{\nu}$	An element of $\mathcal{V} \subseteq \mathbb{Z}_2^{\otimes N}$. The N-orbital occupation vector representing a fermionic basis state, see (2.17).
$\boldsymbol{\omega}$	An element of $\mathbb{Z}_2^{\otimes n}$. A binary vector representing a product state in the n-qubit basis, see (2.18).
$\boldsymbol{p}$	Binary vector function $\mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2^{\otimes N}$ The parity function,used for the parity operators.
$P(j)$	A subset of $[n]$. The parity set of orbital j, see (2.12).
$\mathcal{P}$	Set of single-qubit Pauli operators $\{X,\, Y,\, Z\}$.
$\boldsymbol{q}$	An element of $\mathbb{Z}_2^{\otimes N}$. Binary vector denoting the occupational change of a vector $\boldsymbol{\nu}$ by a term $\widehat{h}_{\boldsymbol{ab}}$.
R	A binary $(N \times N)$ matrix, where the lower triangle (excluding the diagonal) is filled with ones, see (2.13).
θ_{ij}	A function $[N]^{\otimes 2} \mapsto \mathbb{Z}_2$. The discrete version of the Heaviside function, see (2.13).
$\boldsymbol{u_j}$	A binary vector in $\mathbb{Z}_2^{\otimes N}$ or $\mathbb{Z}_2^{\otimes n}$. The j-th unit vector, in which just component j is one.

$U(j)$ A subset of $[n]$ The update set of orbital j, see (2.12).

$\mathcal{U}^{\boldsymbol{a}}$ A linear operator: $\mathcal{L}\left((\mathbb{C}^2)^{\otimes n}\right)$. Update operator with respect to an occupation of $\boldsymbol{a}$, see (2.31) and (2.32).

$\mathcal{V}$ A subset of $\mathbb{Z}_2^{\otimes N}$. The set of all allowed occupation vectors $\boldsymbol{\nu}$, implementing the basis $\mathcal{B}$, see (2.17).

$\mathrm{w_H}\left(\cdot\right)$ A function $\mathbb{Z}_2^{\otimes N} \mapsto [N] \cup \{0\}$. The Hamming weight of a binary vector, which is the sum of its components.

$\mathfrak{X}$ A map $\left(\mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2\right) \mapsto \mathcal{L}\left((\mathbb{C}^2)^{\otimes n}\right)$. The extraction superoperator, which relates binary functions to quantum gates, see (2.19) - (2.27).

$\mathbb{Z}_2$ The set of binary digits: $\{0,\, 1\}$.

2.9 Further work

Tapering qubits off

There is another approach that can help reducing the number of qubits by exploiting symmetries. Let us consider that a qubit Hamiltonian has a set of stabilizers. As stabilizer conditions constrain degrees of freedom, they can be taken into account to eliminate an equivalent number of qubits. In fact, we find that per stabilizer condition eliminated one qubit can be tapered off, which means removing a number of qubits equivalent to the number of stabilizer generators. In [34], where this idea was developed, a quantum algorithm is devised to render the action of the problem Hamiltonian trivial on the qubits to be removed. However, in the wake of [43], we have developed a perhaps simpler method, with which qubits can be tapered off a Hamiltonian H, given only the generating set of stabilizers. While those stabilizers have to take the form of Pauli strings, we might have to simulate the system in the negative subspace of some of them, which effectively corresponds to those stabilizers being defined with a minus sign: $S \in \pm\{\mathbb{I},\, X,\, Y,\, Z\}^{\otimes n}$. Note that those stabilizers might not be a product of natural symmetries. Our original motivation was to test logical Hamiltonians, as they appear in the next chapter. These are Hamiltonians of systems in which stabilizer conditions are defined as part of a quantum code, but like before we can regard the stabilizers as

$\tau\backslash\sigma$	X	Y	Z
$\mathbb{I}$	h	h	h
X	$\pm h\cdot p$	$\pm ih\cdot p$	h
Y	$\mp ih\cdot p$	$\pm h\cdot p$	$\pm ih\cdot p$
Z	h	h	$\pm h\cdot p$

Table 2.3. Removing one qubit from a Pauli string $\tau \otimes h$, by eliminating a stabilizer generator $\pm\sigma\otimes p$. Here p and h are Pauli strings on $n-1$ qubits, and σ, τ are Pauli operators on an isolated qubit. The entries of the table show the $(n-1)$-qubit equivalents of $\tau \otimes h$ for various instances of τ (rows) and σ (columns).

boundary conditions imposed on a physical system of spins. Testing the Hamiltonian spectra, and thus the code space, is therefore not possible unless we somehow include these conditions. At this point, qubit tapering can be used to eliminate all stabilizer conditions and turn a logical Hamiltonian into a physical one. The spectrum of the latter can then be matched with its expectation to verify the subspace.
We will now describe our procedure in detail. It relies on a routine that tapers off one qubit for each stabilizer generator so the following scheme is to be repeated until no stabilizers are left. In the beginning, we isolate one qubit on which the selected stabilizer acts non-trivially as $\sigma \in \{X, Y, Z\}$. Let us write the stabilizer as $S = \pm\sigma \otimes p$, where the first register denotes the isolated qubit, and the second register is comprised of the remaining qubits, on which the stabilizer acts as the Pauli string $\pm p$. We now replace all logical operators (and remaining stabilizers), $\tau \otimes h$, with the entries (corresponding to the concrete instances of τ and σ) in Table 2.3.

Obviously, the isolated qubit has been removed, and the stabilizer S is discarded. Let us prove this method. The idea is that by the stabilizer, a Pauli string $\tau \otimes h$ is re-expressed as either $\mathbb{I} \otimes h'$ or $\pi \otimes h''$, where $\pi \in \{X, Y, Z\}/\{\sigma\}$. In the table, we have have conventionally chosen to fixed $\pi = Z$ in cases $\sigma \in \{X, Y\}$ and $\pi = X$ for the case $\sigma = Z$. This is achieved by multiplying $\tau \otimes h$ with $\sigma \otimes p$ in case $\tau \notin \{\mathbb{I}, \pi\}$. Since now each term is acting on the isolated qubit as either π or $\mathbb{I}$, it can be disregarded from the underlying eigenvalue problem, as it must be in the $\pm$-eigenstate of π. To save qubits, we just replace it with its eigenvalue ±1: $\mathbb{I} \otimes h' \mapsto \otimes h'$ and $\pi \otimes h'' \mapsto \pm h''$. Note that the concrete choice between the two eigenvalues is irrelevant: the case in which the tapering is performed within the (-1) eigenspace produces a Hamiltonian $H \mapsto H^-$, that is related to its $(+1)$ counterpart, H^+, by the unitary transform: $H^+ = p\ H^- p$, which means H^+ and H^- are isospectral.

Term reduction

When applying the tapering, the number of Hamiltonian terms is not increased by the procedure outlined above. However, it can very well decrease. Two Hamiltonian terms, h^0 and h^1, can for instance be related by the condition $h^1 = \pm h^0 \cdot S$. After S is eliminated by tapering, the two terms will not stay distinct, but rather be added or subtracted according to the sign, meaning that at least one or even both terms vanish. Even without eliminating qubits, this feature can be utilized to reduce the number of Hamiltonian terms. This is particularly useful for logical Hamiltonians, since for transforms concatenated with quantum codes, the number of terms tends to proliferate.[2] The problem is that while many of those terms are equivalent up to the multiplication by stabilizers, however they are not identical, such that classical software cannot merge or cancel them. We have defined a classical routine to eliminate redundant terms while maintaining the code space. Like before, we begin by choosing a qubit on which the first stabilizer acts as σ^1, and by multiplication fix each logical term as well as the remaining stabilizer generators to act on it as $\pi^1 \neq \sigma^1$ or $\mathbb{I}$, the identity. However, while discarding the first stabilizer we are not going to remove the selected qubit – a procedure that is repeated for every stabilizers in the list of generators. Note that we always have to select a qubit that is distinct from the ones already fixed. For a total of r stabilizer generators, we end up with each logical Pauli string h transformed into $\lambda^h \otimes \eta^h$, where $\lambda^h \in \bigotimes_{i=1}^{r} \{\mathbb{I},\, \pi^i\}$ is a fixed string on the r qubits selected, and η^h the 'free' remainder of h on the $n - r$ qubits left. When every Hamiltonian string h is brought into this form, redundant terms cancel or merge. However, the Hamiltonian strings probably had an optimized Pauli weight as h, that is likely to be not conserved in $\lambda^h \otimes \eta^h$. To keep the weight optimized, we perform the above procedure within a table, such that in the end each original string h is stored together with its fixed form. We have thus obtained a look-up table with which every remaining string $\lambda^h \otimes \eta^h$ of the Hamiltonian can be mapped back to h, retrieving its optimized weight.

[2] Unpublished observations.

Chapter 3

Embedding simulations with quantum codes

3.1 Background

While small quantum simulations have been performed on few-qubit devices across all platforms [24, 41, 44–48], and efforts are undertaken to scale devices up, the quantum simulation of larger fermionic systems is still a challenge. Critical factors that determine the feasibility of an algorithm would be its qubit requirements, its gate cost (in terms of magic states when error-corrected, and in terms of two-qubit gates when noisy) [28, 49] and circuit depth (a measure of the algorithm run time, where each time step is the duration of one quantum gate). Quantum algorithms are generally to be kept shallow to ensure that they can be run before the qubit system has decohered. It is thus in our interest to decompose the algorithms into many parts that can be run in parallel, i.e. at the same time. Obviously, one can hope for parallelization if the algorithm is comprised of gate sequences that act on subsets of as few qubits as possible and these subsets do not overlap much. Another factor is that actual quantum devices can have geometric limitations which negatively influence the circuit depth. In a practical setting not every qubit can reach every other qubit, i.e. they cannot be entangled with a single two-qubit gate. To entangle distant qubits, it takes additional efforts in gates and time. Thus another criterion for the reduction of the circuit depth is that gate sequences only act on qubits adjacent on a certain connectivity graph. Although this graph depends on the actual quantum device, we can make an educated guess: devices on which surface code can be run, require a square lattice connectivity graph.

Unfortunately, it is nontrivial to embed fermionic problems in those lattices, which opposes shallow-depth quantum simulation. Let us illustrate the exact issue. In order to bring the problem into a form the quantum computer can process, the fermionic modes need to be embedded into a (two-dimensional) lattice structure related to the qubit connectivity graph. After that, a fermion-to-qubit mapping translates the interactions of those system to a qubit Hamiltonian fit to be simulated. It is this last step in which the problem lies, as simulating the interaction between as little as two fermionic modes usually requires gates acting on large subsets of qubits. This is a consequence of the fermionic wave functions being antisymmetric under particle permutations, which causes the interaction of two fermionic modes to also be sensitive to the occupation of seemingly uninvolved modes, turning into gates on the qubits representing them. This is the same issue that prohibits us from describing fermions on (two-dimensional) lattices in terms of bosons, which could be simulated more easily. In fact, the problems are somewhat intertwined considering that those bosonic descriptions can double as fermion-to-qubit mappings. The Jordan-Wigner transform for instance is widely used as a fermion-to-qubit mapping [24, 45, 46, 48] today, but its appearance in 1928 [19] predates the work of Feynman by half a century. The original work of Jordan and Wigner was rather meant to compare fermionic operators to the operators of (hard-core) Bosons, which are easily mapped to $(1/2)$-spins. For our purposes, the spins are immediately identified as qubits, rendering the transform a default for fermion-to-qubit mappings. However, the Jordan-Wigner transform is effectively one-dimensional and exhibits large deficits in the treatment of two-dimensional systems. In particular it fails to map a fermionic lattice model with local interactions (meaning their interaction range is bounded by a constant) to a model of locally interacting spins. In contrast to that, locally interacting spins on a lattice can be mapped to a locally interacting Boson lattice, due to the bosonic wave function not being antisymmetric [50]. While there are tricks and generalizations to circumvent the deficits of the Jordan-Wigner transform [51–54], not all of them are useful for its role in quantum simulation: there is no ultimate choice for a two-dimensional fermion-to-qubit mapping. However, there is a mapping with which locally interacting fermion and qubit lattices can be related: the *Verstraete-Cirac transform* (VCT) [25] also known as *Auxiliary fermion mapping* [29, 55, 56], can be regarded as a manipulation of the Jordan-Wigner transform, in which additional *auxiliary* particles are added, hence the name. Other works on fermion-to-qubit mappings [27, 29, 37, 57] are based on two transforms proposed by Bravyi and Kitaev in [26]. First, there is the already mentioned Bravyi-Kitaev transform, that, compared to the Jordan-Wigner transform, exhibits an up to exponential improvement on the number of qubits that each fermionic interaction term acts on. The Bravyi-Kitaev transformation however demands a qubit connectivity that is higher than what a square lattice can offer. Second, the mapping referred to as *'Superfast simulation of fermions on a graph'* (BKSF) has the power to map local fermion lattices to local qubit lattices, but the square lattice connectivity is generally only sufficient when the underlying model is an interacting square lattice as well: to make interactions local, the mapping requires a qubit connectivity graph set by the Hamiltonian. When the given connectivity turns into a limitation, classical tools like sorting networks might be applied [58]. Most notably, there are recent attempts to incorporate swapping networks into the fermion-to-qubit mapping. With so-called fermionic swaps [26], not only qubits are swapped but also fermionic modes, in the sense that swapping operations can change the locality of their interactions in the Jordan-Wigner transform. This effectively eliminates the contribution of the fermion-to-qubit mapping to the gate cost and algorithmic depth which is then dominated by the swapping network alone [30, 59].

In this chapter, we want to abstain from swapping and sorting networks to make use of the (two-dimensional) geometric proximity of qubits inside the quantum device. In this way, the gate cost is determined by the range of interactions on the fermionic lattice and distant interactions can be simulated in parallel. For this purpose, we define two-dimensional (nonperturbative) fermion-to-qubit mappings that generalize the Jordan-Wigner transform on the square lattice. We here not only demand that local Hamiltonians of fermions are mapped to local qubit Hamiltonians but want to go beyond nearest neighbor interactions. The exchange interaction between two (distant) modes should involve only the two qubits that these modes correspond to, and some chain of qubits that connects them geometrically. This means that when we imagine the system as a fermion lattice with dimension $(\ell_1 \times \ell_2)$, we want an interaction term of any two modes to transform into a term acting on $O(m)$ qubits, when the modes have a Manhattan distance of m. As a consequence, we can bound the weight of the largest terms by $O(\ell_1+\ell_2)$, rather than $O(\ell_1\times\ell_2)$ as in the case of the Jordan-Wigner transform. In this way the entire simulation only considers operators acting on the shortest possible strings along adjacent qubits, fostering parallelization.

3.2 Results

In this chapter, we introduce a class of fermion-to-qubit mappings, that are two-dimensional generalizations of the Jordan-Wigner transform on a $\ell_1\times\ell_2$ lattice of fermionic sites. The *Auxiliary qubit mappings* (AQMs) are based on the (one-dimensional) Jordan-Wigner transform, concatenated with specific quantum (stabilizer) codes. Stabilizer codes, which play an important role in quantum error correction, encode a logical basis of 2^N degrees of freedom (here $N = \ell_1 \times \ell_2$) in a subspace of a larger system with $n > N$ qubits. The degrees of freedom left are constrained with so-called stabilizer conditions, which means there are $n - N$ (independent) qubit operators $\{S_i\}_i$ that stabilize this basis, i.e. in the logical subspace the expectation value of all stabilizers is one, $\langle S_i \rangle = 1$. In our case, the logical basis encoded is the one of the Jordan Wigner transform, to which $r = n - N$ auxiliary qubits have been added and constrained. The entire procedure is illustrated in Figure 3.1, where the AQM performs the transition from layer (a) to (c), effectively avoiding the nonlocal interactions on layer (b). The codes used for AQMs are planar on the square lattice, and we devise a unitary quantum circuit that switches in between the layers (b) and (c). This circuit has an algorithmic depth that scales with ℓ_1, the length of one of the lattice sides. There is no such operation for mappings found in prior works, the Verstraete-Cirac transform and Superfast simulation. To compare them with the AQMs, we modify the VCT and BKSF, rendering them planar codes with the Manhattan-distance property. The contributions of this chapter

- We introduce three types of Auxiliary Qubit Mappings, each requiring a different amount of auxiliary qubits. Our main result of this paper is the *square lattice AQM*, which uses $2N-\ell_1$ qubits in total. Note that in general, mappings with more auxiliary qubits will in some sense deal better with the second dimension, but none of the mappings generalizing the Jordan-Wigner transform has a total qubit number exceeding $2N$. However, one might be interested in

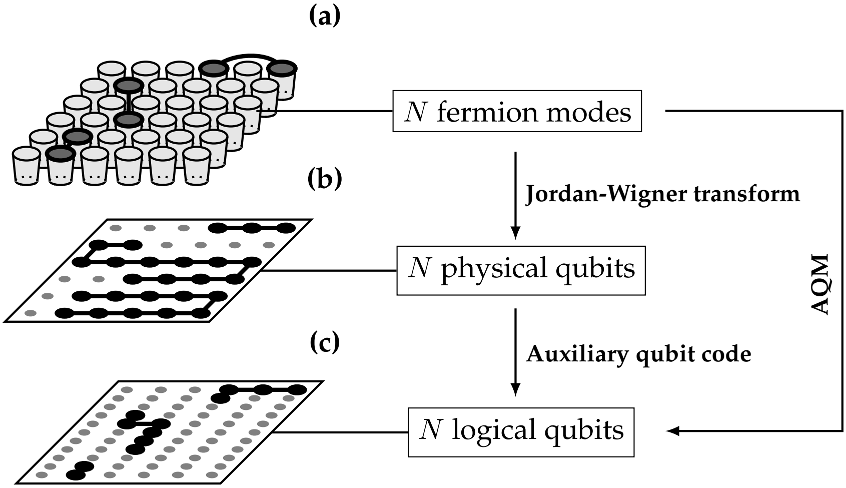


Figure 3.1. Visualizing an Auxiliary Qubit Mapping (**AQM**) as a concatenation of the Jordan-Wigner transform and a particular quantum code. The three layers represent the lattices of fermions and qubits. We have highlighted the same three exchange terms on each lattice, so their transformation can be observed. **(a)** The starting point: a fermionic lattice or two-dimensional embedding of a fermion system with $\ell_1 \times \ell_2$ modes. The three (local) interactions highlighted are brought via the Jordan-Wigner transform onto the (data) qubit layer. **(b)** The data qubit layer, in which two of the formally local interactions now assume a nonlocal form. To restore locality, we need to define a quantum code on the data qubits register and some auxiliary qubits, added to the next layer. **(c)** The final layer: a composite system of n qubits, where we have placed $n - N$ auxiliary qubits in between the data qubits. By the Auxiliary Qubit code, interactions that were local in the top layer can now be made local again. Note also that the interaction in the center of the lattice, which has involved many qubits in the middle layer, is now reduced to act on few qubits again by the Manhattan-distance property.

using fewer auxiliary qubits: this can be the case for instance when simulating lattice models, where we would like to make the physical lattice as large as possible and 'being on a fixed qubit budget' accept a trade-off between circuit depth and the number of auxiliary qubits. A qubit-economic version of this mapping would be the *sparse AQM*, which introduces the parameter $\mathcal{I}$ to regulate the trade-off. Furthermore, with adding only a few qubits we can already obtain a modified version of this mapping which has easy-to-prepare logical states and is called *E-type AQM*. A comprehensive list of all considered fermion-to-qubit mappings, that allows us to compare their properties, is compiled into Table 3.1. For all Auxiliary Qubit Mappings, we provide the initialization circuits of $O(\ell_1)$ depth.

- We demonstrate the Auxiliary Qubit Mappings on the Fermi-Hubbard model, decreasing its algorithmic depth from being linear with the number of data qubits, $O(N)$, to being constant, $O(1)$. This is an important step towards making its simulation scalable (at the expense of more qubits). Lattice models are in general not just interesting by themselves, but also test on how a fermion-to-qubit mapping deals with the second dimension, i.e. the criteria mentioned in the introduction, in a minimal fashion. We explicitly show how the mappings transform the Fermi-Hubbard model into a model of local qubit interactions on the lattice.

- We compare our work, the Auxiliary Qubit Mappings, to the Verstraete-Cirac transform [25] and the Superfast simulation [26] from the literature. As indicated above, we adjust the latter two slightly to make all three mappings comparable. Advantages and disadvantages of each mapping eventually lead us to conclude which of them to recommend for different situations.

While these contributions are covered in Sections 3.5, 3.6 and 3.7, the rest of the paper is organized as follows: in Section 3.3, we provide a more structured introduction to the layout of the quantum device and the established fermion-to-qubit mappings. We discuss criteria for a 'good' mapping in detail and that the Jordan-Wigner transform has deficits in those regards. In Section 3.4, we illustrate the effect of quantum codes, such as the ones that are the blueprint for the AQMs, on a given Hamiltonian. While the AQMs are an original idea, we cannot claim the same about their theoretical backbone: the foundations for Auxiliary Qubit codes are basically used in [60], although there the stabilizer formalism was not employed. As a consequence, one auxiliary qubit would have to be added for each term in the Hamiltonian, which is a large overhead that can be avoided by using the underlying principle to define quantum codes. We derive these codes from scratch in Section 3.9.1. Some minor contributions are provided outside the main text of this chapter. In Section 3.9.2, we study the class of tree-based mappings, to which the Bravyi-Kitaev transform belongs. The Bravyi-Kitaev transform itself does not do well with the square lattice, but we provide a general method to tailor and embed similar mappings to arbitrary two-dimensional setups. Section 3.9.3 is mostly providing details on the Verstraete-Cirac transform and Superfast simulation, but we also tackle some side issues by deriving the logical basis of both mappings.

	Jordan-Wigner (S-pattern)	Verstraete-Cirac transform	Superfast simulation	Square lattice AQM	E-type AQM	Sparse AQM
Origin	[19]	[25]	[26]	[43]	[43]	[43]
Aux. qubits	0	$\ell_1\ell_2$	$\ell_1\ell_2 - \ell_1 - \ell_2$	$\ell_1\ell_2 - \ell_1$	ℓ_2	$(\ell_2 - 1)(\frac{\ell_1 - 1}{\mathcal{I}} + 1)$
String length (general)	$O(\ell_1\ell_2)$	$O(2\ell_1 + \ell_2)$	$O(2\ell_1 + 2\ell_2)$	$O(\ell_1 + 2\ell_2)$	$O(2\ell_1 + \ell_2)$	$O(\ell_1 + 2\ell_2)$
Manhattan-distance property?	✗	✓	✓	✓	✗	approximately
String length (lattice)	$O(\ell_1)$	$O(1)$	$O(1)$	$O(1)$	$O(\ell_1)$	$O(\mathcal{I})$
Simulation time (lattice)	$O(\ell_1\ell_2)$	$O(1)$	$O(1)$	$O(1)$	$O(\ell_1\ell_2)$	$O(\mathcal{I}^2)$
+ cancellations	$O(\ell_1)$	$O(1)$	$O(1)$	$O(1)$	$O(\ell_1)$	$O(\mathcal{I})$
Restores locality?	✗	✓	✓	✓	✗	approximately

Table 3.1. All fermion-to-qubit mappings discussed in this work. We consider a $N = (\ell_1 \times \ell_2)$ square lattice block of fermionic modes, and compare the number of auxiliary qubits, or more generally the total number of qubits minus N. We also compare the scaling of the number of qubits involved in two types of Hamiltonians: generic ones, in which we expect interactions between every mode, and lattice models, with only nearest-neighbor interactions. For the former, we also ask whether long-range interactions can be mapped to operators involving qubits along a direct path (Manhattan-distance property). For the lattice models, we specify the expected algorithmic depth for simulating the entire Hamiltonian by e.g. Trotterization and whether their locality is restored after the transformation. Note that the simulation time is obtained using simulation gadgets that adhere to the square lattice connectivity of the qubits, however, we take into account that some simulation algorithms allow for partial cancellation of overlapping Pauli strings in the Hamiltonian. Note also that $\mathcal{I}$ is a parameter of the last mapping that can be chosen as some integer number: $1 \leq \mathcal{I} \leq \ell_1 - 1$. This parameter determines how well the Manhattan-distance property and locality is approximated.

3.3 Preliminaries

In this section, we describe the influence of fermion-to-qubit mappings on the algorithmic depth of quantum simulation in a setup of square-lattice qubit-connectivity. In particular, we will discuss criteria which render mappings 'good' in the sense that they allow for parallelization and low gate costs. For that purpose, we will give a theoretical description of the qubit layout and sketch the simulation algorithms. Let us start however by stating the role of fermion-to-qubit mappings for quantum simulation in general. We generally advise the reader familiar with the subject to skip ahead to Section 3.4, and if necessary use the table of notations offered in Section 3.10.

The goal of quantum simulation is to approximate the ground state and the ground-state energy of a given Hamiltonian. When the Hamiltonian acts on a space of fermions, a fermion-to-qubit mapping serves as translator between the quantum system to be simulated and the qubit system inside the quantum computer. That not only entails a correspondence of basis states, but also a transformation of the Hamiltonian. The Hamiltonian after its transformation with the mapping, is henceforward acting on the qubits inside the quantum computer. We here consider the case where the qubit system underlies architectural constraints, that we want to abstract with the following model.

Our setup is a two-dimensional quantum device that we describe with a planar graph, where each of the n vertices is a qubit. In this model, it is assumed that we can individually and simultaneously perform Pauli-rotations on every single qubit. However, entangling gates can only be applied between two qubits that share an edge in the graph. We assume that we can perform two-qubit gates individually per edge, but qubits involved in one gate cannot be part in another at the same time. Although we do not want to specify which kind of two-qubit gate is native to the quantum device, we want to assume that we can do CNOT-gates in $O(1)$ time using only a few native gates. The full qubit connectivity graph will furthermore be assumed to be a square lattice, so we can only perform entangling gates between qubits that are nearest neighbors, see Figure 3.2(a). Note that the individual connectivity graphs, that every fermion-to-qubit mapping in this chapter comes with, are subgraphs of Figure 3.2(a), such that every mapping can be embedded in the considered qubit system.

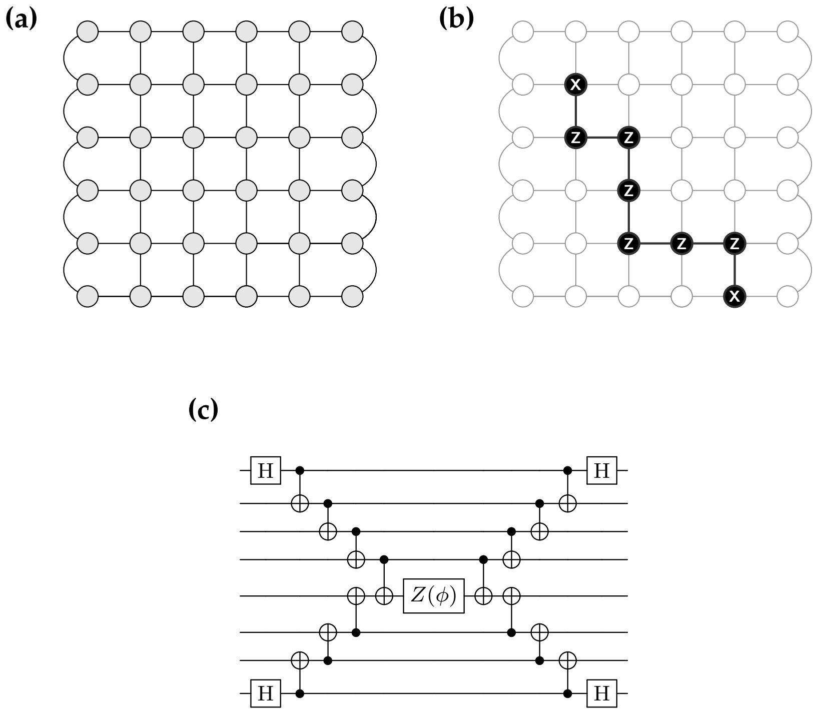


Figure 3.2. Simulation of Pauli strings in a system with limited connectivity. **(a)** Qubit connectivity graph: the vertices are qubits. Two-qubit gates can be performed only between qubits coupled by an edge. **(b)** Simulating some Pauli string $(X \otimes Z^{\otimes 6} \otimes X)$ on the quantum device: the qubits involved, and the edges along which entangling gates are performed, are highlighted. Inscriptions X, Y and Z indicate which Pauli operator acts on each qubit. **(c)** Simulating a Pauli string, here we simulate the propagator $\exp(i\,\phi\, X \otimes Z^{\otimes 6} \otimes X)$, where ϕ is an angle. The Pauli string could be the one in (b). In general, this circuit stores the parity information of the involved qubits on one of them, which is done by chains of CNOT-gates. The inscriptions X, Z and Y determine for each individual qubit whether it is in the Hadamard, computational or Y-basis in the process. Note that it does not play a role on which of the qubits the parity of the others is collected, but to optimize the simulation time, a qubit in the middle of the chain is chosen. On that qubit the phase rotation $Z(\phi) = \exp(i\,\phi\, Z)$ is performed, after which the chains are uncomputed.

3.3.1 Simulating a qubit Hamiltonian

In order to elucidate the connection between the mapping and the depth and cost of the simulation algorithms, we need to understand these algorithms better. Let us assume the fermion-to-qubit mapping transforms a Hamiltonian into the form of Pauli strings, i.e. the sum $H = \sum_h \Gamma^h \cdot h$, where $\{\Gamma^h\}$ are real coefficients associated to a Pauli string on n qubits, $h \in \{X, Y, Z, \mathbb{I}\}^{\otimes n}$. Note that we will refer to the number of qubits, that a string h acts on nontrivially, as (operator) weight and (string) length, interchangeably.

Quantum simulation algorithms have different ways to search for the ground state of H. Depending on which algorithm is used, the Pauli strings h have to be either measured, or their propagator simulated (conditionally) [5, 6]. With a propagator we mean the operator $\exp(i\,\phi\,h)$, where ϕ is an angle that typically is some function of Γ^h. Using CNOT-gates, we simulate such a propagator with the gadget like in Figure 3.2(c), where chains of these gates copy parity information across the lattice onto a single qubit, on which then a Z-rotation around the angle ϕ is performed and afterwards the CNOT-chain is uncomputed. For quantum eigensolvers, this qubit will be measured instead. Often we need the rotation to be conditional on the state of another qubit, so conventionally the Z-rotation, $Z(\phi) = \exp(i\,\phi\,Z)$, is to be replaced with a controlled rotation, $\mathbb{I} \otimes |0\rangle\langle 0| + Z(\phi) \otimes |1\rangle\langle 1|$ where the first qubit is the one that holds the parity information, and the second is the control, typically an auxiliary qubit of a phase estimation procedure. Alternatively, the quantum phase estimation algorithm can be adapted to include control qubits in the string, namely to simulate the propagator $\exp(-i\,\frac{\phi}{2}\,h \otimes Z) = \exp(-i\,\frac{\phi}{2}\,h) \otimes |0\rangle\langle 0| + \exp(i\,\frac{\phi}{2}\,h) \otimes |1\rangle\langle 1|$ instead.

For phase estimation-based algorithms, the propagator of the entire Hamiltonian, $\exp(iH\phi)$ needs to be simulated, which invokes the propagator of each string at least once (e.g. [61, 62]). Other algorithms invoke each string multiple times: Trotterization [8, 9] approximates the Hamiltonian propagator as repeating sequences of all string propagators $\exp(i\,\phi\,h)$, and in *iterative phase estimation* [23], a repeated application of $\exp(iH\phi)$ increases the accuracy of the computed energy. In general, H does not even have to be a Hamiltonian: it could also be an operator that prepares a trial state with *Givens rotations* [30] or implements a *unitary coupled-cluster* operator [63]. In any case, we will expect there to be a large number of strings in H so we would like to apply the gadgets 3.2(c) in parallel to keep the simulation shallow whenever possible. Let us coordinate the simulation of all those propagators by switching to layout diagrams like the one in Figure 3.2(b), instead of using circuit diagrams like in panel (c). This gives us an idea of all the qubits involved and how they are coupled, but leaves out certain details about for instance the specific simulation algorithm. Our ability to parallelize the simulation is determined by the fermion-to-qubit mapping, in particular in the shape of the strings that it outputs. In regard of our connectivity setup 3.2(c), we consider a fermion-to-qubit mapping as good, if it outputs Hamiltonians H with Pauli strings that are *short*, *continuous* and *non-overlapping*. We will now explain these criteria:

short - The length of a Pauli string is the number of qubits that it acts on nontrivially. While the gadget in Figure 3.2(c) implements a propagator in a number of time steps that scales linearly with the amount of qubits involved, other implementations have been conceived. As can be seen in [39, 64], the gadget can be replaced with one that performs the same operation with an up to exponential improvement in the circuit time, so at most $O(\log n)$. However, taking into account the (limited) qubit connectivity of the square lattice, we want to stick to the gadget of Figure 3.2(c). Although a time reduction can be achieved for Pauli strings acting on a nonlinearly distributed subset of qubits, we generally expect a time scaling linear in the string length. As the number of time steps is interchangeably connected to the circuit depth, we have an interest in keeping the Pauli strings as short as possible.

continuous - In general, Pauli strings in H will not only act on nearest neighbors, this means we cannot connect the qubits involved along shared edges as it is done in Figure 3.2(b). Connectivity problems are symptomatic for layouts like this, in which only nearest-neighbors are coupled. Let us assume that two qubits need to be connected in a gadget like 3.2(c), but they do not share an edge and the shortest path along edges encompasses a number of m uninvolved qubits. In order to skip these qubits, $O(m)$ additional two-qubit gates and time steps are required. In case the native two-qubit gates are either iSwap or $\sqrt{\text{Swap}}$, the outer qubits can be connected by a chain of Swap gates, which costs $2m$ native gates in the former case and $4m$ in the latter. For systems with native CNot-gates the formation Swap gates with three CNots is unnecessarily expensive, so instead we amend gadgets like in Figure 3.2(c) with a construction that includes the m inner qubits in the CNOT-chains, but compensates for their contribution. We present two versions of such a compensation circuit in Figure 3.3, where the left panel shows us the gate that we would like to perform but cannot: we would like the configuration of the first qubit to be added to the last qubit by a nonlocal CNOT-gate. In the end, the circuits in the center and on the right achieve that task but render the m uninvolved qubits useless until the circuit is uncomputed. The additional cost in time and gates is $4m$, which means that it is cheaper to include a qubit in a string than to skip it. In conclusion, compensating or swapping of qubits is possible, but we would prefer to avoid the additional cost and rather deal with continuous strings.

non-overlapping - The overlap of two (or more) Pauli strings is the number of qubits in the intersection of the sets of qubits the strings act on. Two Pauli strings that are both acting nontrivially on a common subset of qubits are hard to simulate in parallel, as these qubits get parity information attached to them like in Figure 3.2(c). Unless these qubits are located at the beginning of a chain or if one string is a substring of the other, this parity would have to be corrected for. Later, we will briefly discuss the possibility of gate cancellations between similar, overlapping strings. While this has been suggested for Trotterization in [39], its impact on the approximation error is not well understood yet. Product formula approaches based on coalescing or randomization offer little or no choice in the term ordering [49, 65–67]. Thus, avoiding the need for cancellations, we ideally would like our mapping to transform all pairs of commuting fermionic operators into non-overlapping Pauli strings.

3.3.2 S-pattern Jordan-Wigner transform

Based on the insights of the previous sections, we will now review what is probably the standard fermion-to-qubit mapping [19]. In case of the Jordan-Wigner transform, the transformation matrix A can be regarded as the identity: $A = A^{-1} = \mathbb{I}$. From (2.14), we derive the number operators

$$c_j^\dagger c_j \mathrel{\hat{=}} \frac{1}{2}\left(\mathbb{I} - Z_j\right) \tag{3.1}$$

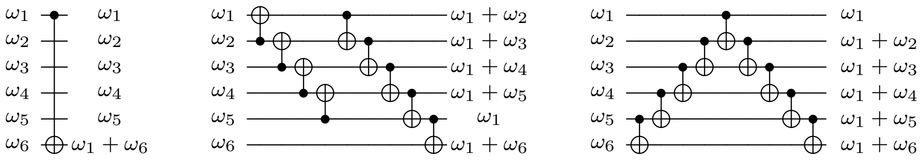

ω_1 ω_1
ω_2 ω_2
ω_3 ω_3
ω_4 ω_4
ω_5 ω_5
ω_6 $\omega_1 + \omega_6$
ω_1 $\omega_1 + \omega_2$
ω_2 $\omega_1 + \omega_3$
ω_3 $\omega_1 + \omega_4$
ω_4 $\omega_1 + \omega_5$
ω_5 ω_1
ω_6 $\omega_1 + \omega_6$
ω_1 ω_1
ω_2 $\omega_1 + \omega_2$
ω_3 $\omega_1 + \omega_3$
ω_4 $\omega_1 + \omega_4$
ω_5 $\omega_1 + \omega_5$
ω_6 $\omega_1 + \omega_6$

Figure 3.3. Skipping several qubits in a CNOT-chain. Here we consider the effect of the circuits on a computational basis state $(\bigotimes_i |\omega_i\rangle)$, mapping it to a state $(\bigotimes_i |\omega_i'\rangle)$. We denote the qubit values ω_i and ω_i' on the left and right side of each circuit. **Left:** The desired circuit, a CNOT-gate that adds the parity from the first qubit to the last. For connectivity reasons, this gate is not possible: we can only connect adjacent qubits. **Center/Right:** Two circuits in which the middle qubits are compensated for in order to entangle the first and last qubit. To get rid of the effect on qubits 2 - 5, the gadgets have to be partially uncomputed, but in propagators like in Figure 3.2(c), this is not necessary.

and hopping terms (for $i < j$)

$$\begin{aligned} \mathrm{h}_{ij}\, c_i^\dagger c_j + (\mathrm{h}_{ij})^*\, c_j^\dagger c_i \quad &\hat{=} \quad \frac{1}{2}\,\mathrm{Re}(\mathrm{h}_{ij}) \left(\bigotimes_{k=i+1}^{j-1} Z_k \right) (X_i \otimes X_j + Y_i \otimes Y_j) \\ &\quad + \frac{1}{2}\,\mathrm{Im}(\mathrm{h}_{ij}) \left(\bigotimes_{k=i+1}^{j-1} Z_k \right) (Y_i \otimes X_j - X_i \otimes Y_j)\ . \end{aligned} \tag{3.2}$$

While the number operator is transformed into just a constant term and a term that acts on one qubit only, the hopping terms are transformed into a string that exhibits long substrings of Z-operators, $(\bigotimes_{k=i+1}^{j-1} Z_k)$, sometimes called parity (sub-)strings. The right-hand side of (3.2), which describes an interaction of the fermionic modes i and j, translates into several strings with X- and Y-operators on the corresponding qubits of i and j, and all qubits of indices k, with $i < k < j$, are part of the parity substring. Although the parity string does us the service of connecting the qubits i and j in that way, it is also the reason that Pauli strings produced by the Jordan-Wigner transform are of length $O(N)$.

While the nature of our problem determines the Hamiltonian coefficients (such as h_{ij}) with respect to the fermionic wave functions, it is up to us to label each fermionic mode such that we minimize the appearance of long

Pauli strings in H. While problems that are intrinsically one-dimensional can be mapped to local Hamiltonians, long strings can generally not be avoided for systems in higher spatial dimensions.

The question is how to incorporate the Jordan-Wigner transform into the square lattice layout. There is a natural solution: given a $N = (\ell_1 \times \ell_2)$-matrix of qubits, we need to use only $N - 1$ edges to connect them in canonical order like beads on a string, see Figure 3.4(a). Due to the windings of the pattern on the block boundaries, we will refer to this particular way of using the Jordan-Wigner transform on a square lattice as *S-pattern Jordan-Wigner transform*. Let us now describe its properties in order to assert how good a mapping it is. The mapping produces strings that are *continuous*: although arbitrary terms (like $c_i^\dagger c_j^\dagger c_k c_l$) will in general not be transformed into continuous Pauli strings, creation/annihilation operator pairs $c_i^\dagger c_j$ will. Unfortunately the resulting Pauli-strings are neither *short* nor *non-overlapping*. As the parity strings encompass all the qubits in between i and j, the string can even span several rows, see Figure 3.4(b). This leads not just to a high gate count and algorithmic depth, but also occupies a large portion of qubits at once, effectively hindering parallelization.

Let us consider an illustrative example: if we want our quantum device to simulate a two-dimensional lattice of sites with fermionic occupation and nearest-neighbor hopping, we encounter two kinds of terms. Short ones, where the exchange between nearest-neighbors $c_i^\dagger c_{i+1} +$ h.c. yields the Pauli strings $(X_i \otimes X_{i+1} + Y_i \otimes Y_{i+1})/2$, and long ones, as the nearest-neighbor hoppings in the vertical direction will result in strings that can be seen in Figure 3.4(c). Although these are nearest-neighbor interactions, they use all qubits around the winding linking the two rows, so all vertical hopping terms between two sites in the same two rows will overlap. The S-pattern Jordan-Wigner transform thus has the property to transform operators, that are geometrically local in second quantization into nonlocal Pauli strings on the lattice. In Section 3.6, we will learn that it is those vertical hopping terms, that prevent us from simulating lattice models efficiently.

The verdict for the S-pattern Jordan-Wigner transform is that it is not good in the sense of our criteria, but good enough to serve as a foundation for better mappings. In the following, we will introduce mappings modifying the Jordan-Winger transform in using quantum codes to cancel nonlocal parity strings, which will make the resulting strings short

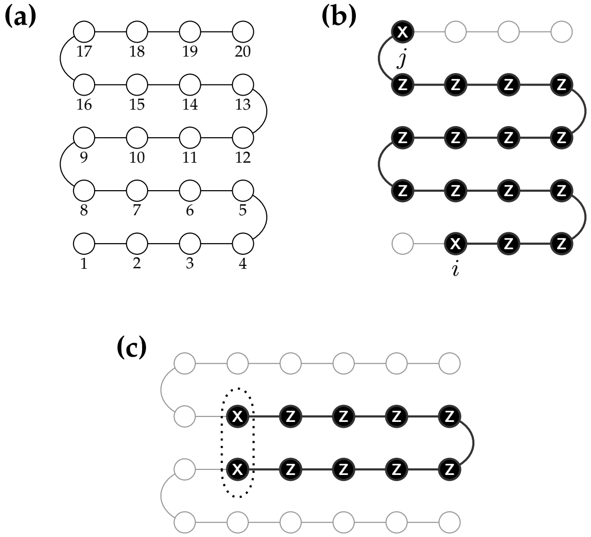


Figure 3.4. **(a)** The connectivity graph for the S-pattern Jordan-Wigner transform. **(b)** Simulating a Pauli string $(X_i \otimes Z_{i+1} \otimes \cdots \otimes Z_{j-1} \otimes X_j)$, that can be considered half of a hopping term. The string is highlighted on the device in the same way as in Figure 3.2(b). **(c)** Simulation of a Pauli string associated with a fermionic hopping between the two encircled qubits (dotted line). The hopping is in the vertical direction (diagonal to the S-pattern) which unfortunately involves gates on all qubits on the S-pattern between the two qubits.

and non-overlapping. This will lead to a certain overhead in auxiliary qubits, placed along with the original $(\ell_1 \times \ell_2)$-block of data qubits on a square lattice. In contrast to the S-pattern Jordan-Wigner transform, the mappings to follow embrace the second dimension as a useful tool.

Note that there are other alternatives to the Jordan-Wigner transform. The Bravyi-Kitaev transform [26, 27, 29, 37] is known to produce Pauli strings of weight $O(\log N)$ instead of $O(N)$. For $N > 16$ it can however be rather difficult to embed the mapping into a square lattice such that it outputs continuous strings. For a geometric interpretation of the Bravyi-Kitaev transform and related mappings we would like to refer the reader to Appendix 3.9.2.

3.4 Techniques

3.4.1 Motivation

Here we motivate the general concept of Auxiliary Qubit Mappings. The starting point will be a nonlocal Hamiltonian obtained by transformation with some linear mapping from Section 2.3. We then define quantum codes in order to restore operator locality. These codes will act on the original system extended by several 'auxiliary' qubits. The effect of such codes on the Hamiltonian will be studied.

Consider that we have an N-qubit Hamiltonian H_{dat},

$$H_{\mathrm{dat}} \ = \ \sum_{h \in \mathcal{S}} \Gamma^h \cdot \ h_{\mathrm{dat}} \, , \tag{3.3}$$

where $\mathcal{S}$ is the set of all Pauli strings occurring in the Hamiltonian, $\mathcal{S} \subseteq \{X, Y, Z, \mathbb{I}\}^{\otimes N}$ with all Γ^h being real, non-zero coefficients. Let us omit the qubit subscripts for now. Although we want to remain fairly general at this point, the reader can already think of (3.3) as the result of a Jordan-Wigner-transformed Hamiltonian (1.8). In general, the problem with this Hamiltonian is that $\mathcal{S}$ contains variations of Pauli strings that are either too long, discontinuous or otherwise inconvenient to us. Thus we would like to somehow replace these strings inside the Hamiltonian, even if it means that we need to add qubits to the system. Let us first consider a naïve approach which indicates the challenges of the method. We then tackle these challenges with a more sophisticated proposal. For the moment, let there be for exactly one inconvenient string $p \in \{X, Y, Z, \mathbb{I}\}^{\otimes N}$, that either appears in the Hamiltonian directly, or is the nonlocal substring of some Hamiltonian strings $\{h'\} \subset \mathcal{S}$. To bring the Hamiltonian in a convenient form, we would like to multiply every such string h' with p. Now we entangle an additional qubit to the system. Ideally, we would like to find the Pauli operator $\sigma \in \pm\{X, Y, Z\}$, acting on the added qubit, such that for every state $|\varphi\rangle$ on the original system of N qubits, there exists a state $|\widetilde{\varphi}\rangle$ on the system extended by the $(N+1)$-th qubit, on which H has the same effect as on $|\varphi\rangle$, but $(p \otimes \sigma)$ is a stabilizer:

$$(p \otimes \sigma) \, |\widetilde{\varphi}\rangle = |\widetilde{\varphi}\rangle \qquad \text{implying} \quad (p \otimes \mathbb{I}) \, |\widetilde{\varphi}\rangle \ = \ \left(\mathbb{I}^{\otimes N} \otimes \sigma\right) |\widetilde{\varphi}\rangle \ . \tag{3.4}$$

If this was true, then every time p appears as a string in the Hamiltonian we could just replace it with σ, or multiply inconvenient strings $(h' \otimes \mathbb{I})$

by $(p \otimes \sigma)$ to cancel the nonlocal substrings. However, this is generally not possible: when there are terms in $\mathcal{S}$ that anticommute with p, then H will destroy the stabilizer state $|\widetilde{\varphi}\rangle$. This means that the state is altered in a way that (3.4) is no longer valid. The simulation of the adjusted Hamiltonian on such a broken stabilizer state subsequently no longer describes the correct time evolution of the underlying N-qubit system. We thus need to adjust the Hamiltonian $H \to H^{(\kappa)}$, where $H^{(\kappa)}$ generally acts on $N + 1$ qubits even without having its terms multiplied by stabilizers yet. This has to be done in a way as to ensure that the time evolution of $|\widetilde{\varphi}\rangle$ according to $H^{(\kappa)}$ can be mapped back to the time evolution of $|\varphi\rangle$ according to H. At the same time we need to demand $[H^{(\kappa)}, p \otimes \sigma] = 0$ and that $(p \otimes \sigma)$ is a stabilizer like in (3.4). Only then we can use $(p \otimes \sigma)$ to cancel p inside the terms of $H^{(\kappa)}$, and so obtain a convenient Hamiltonian $\widetilde{H}$.

We now refine our approach accordingly, considering also the appearance of multiple strings p (and picking up qubit subscripts as well). In H_{dat}, we identify r Pauli strings p^i_{dat} (for $i \in [r]$) that we would like to cancel as we have done with a single string p above. Furthermore, we would like to have the option for every Hamiltonian term h_{dat} to multiply it with either several, one or none of the strings $\{p^i_{\text{dat}}\}$. This is done by repeating the above procedure for each of the r strings. To that end, we add r qubits to the system: grouping them together we introduce the r-qubit auxiliary register aux $= \{N+1, N+2, \ldots, N+r\}$. We assume that at the beginning, the aux-register is initialized in the state $|0^r\rangle = |0\rangle^{\otimes r}$. Our goal is to cancel the i-th string p^i_{dat} with a single Pauli operator on the $(N + i)$-th qubit: σ^i_{N+i}. Thus we need to find a unitary quantum circuit which entangles the aux-register with the data qubits in a certain way: it has to implement a unitary $V_{\text{aux dat}}$, such that for every state $|\varphi\rangle_{\text{dat}}$ (1.5), we have a state in the composite system, $|\widetilde{\varphi}\rangle_{\text{aux dat}}$ with

$$\begin{aligned} V_{\text{aux dat}}\, |\varphi\rangle_{\text{dat}} \otimes |0^r\rangle_{\text{aux}} &= |\widetilde{\varphi}\rangle_{\text{aux dat}} \\ \text{and} \qquad (p^i_{\text{dat}} \otimes \sigma^i_{N+i})\, |\widetilde{\varphi}\rangle_{\text{aux dat}} &= |\widetilde{\varphi}\rangle_{\text{aux dat}}\,, \end{aligned} \tag{3.5}$$

for all $i \in [r]$. To make this work even on a conceptual level, we need to demand that all p^i_{dat} commute pairwise, otherwise there cannot be a common stabilizer state of all $(p^i_{\text{dat}} \otimes \sigma^i_{N+i})$. Once the stabilizer state is obtained, we maintain it by adjusting every term of Hamiltonian (3.3) with a Pauli string on the auxiliary register. This is done in a way such that the action of the adjusted term on the enlarged system is the same as the action of the original term on the original system. The adjustments are:

$$\begin{aligned} h_{\text{dat}} &\mapsto (h_{\text{dat}} \otimes \kappa^h_{\text{aux}}) \quad \text{with} \\ V^\dagger_{\text{aux}\,\text{dat}} \, (h_{\text{dat}} \otimes \kappa^h_{\text{aux}}) \, |\widetilde{\varphi}\rangle_{\text{aux}\,\text{dat}} &= h_{\text{dat}} \, |\varphi\rangle_{\text{dat}} \otimes |0^r\rangle_{\text{aux}} \, , \end{aligned} \tag{3.6}$$

where κ^h_{aux} is the Pauli substring on the auxiliary register that is correcting h_{dat}. Note that in case h_{dat} already commutes with all the stabilizers, κ^h_{aux} is the identity. Of course we would like the above relation to hold for every string in the Hamiltonian, $h_{\text{dat}} \in \mathcal{S}$, but as we have effectively defined a quantum code encoding the entire Hilbert space of the N data qubits, h_{dat} can be an arbitrary N-qubit Pauli string. Now by virtue of the stabilizer conditions (3.5), we can multiply the adjusted terms $(h_{\text{dat}} \otimes \kappa^h_{\text{aux}})$ by any of the operators $(p^i_{\text{dat}} \otimes \sigma^i_{N+i})$, and thus get rid of their detrimental parts. The resulting logical operators $\widetilde{h}_{\text{aux}\,\text{dat}}$ define a convenient (logical) Hamiltonian

$$\widetilde{H}_{\text{aux}\,\text{dat}} = \sum_{h \in \mathcal{S}} \Gamma^h \cdot \widetilde{h}_{\text{aux}\,\text{dat}} \, . \tag{3.7}$$

3.4.2 Definitions

Generally, the auxiliary qubits can be added in the computational basis to cancel strings $p^i_{\text{dat}} \in \{\mathbb{I},\, Z\}^{\otimes N}$ with Z-operators $\sigma^i_{N+i} = Z_{N+i}$. As an enhancement of the Jordan-Wigner transform, codes like this can be used to cancel nonlocal parity strings. The adjustment strings (of a term h_{dat}) κ^h_{aux} would then for all $k \in [r]$ contain X_{N+k} if h_{dat} anticommutes with p^k_{dat}. Note that the codes defined in this way (with only Z-stabilizers) have the property to map N-qubit computational basis states to states in the computational basis on n qubits, a trait that is useful for state preparation. These codes however have their limitations, as they can easily demand adjustment strings κ^h_{aux} of weight $O(r)$.

Other schemes specifically minimize the weight of κ^h_{aux} . The methods of Subaşı and Jarzynski [60] effectively define codes with auxiliary qubits in Hadamard basis that allow for an arbitrary choice of Pauli strings p^i_{dat}, as long as all r strings commute pairwise. The p-strings are subsequently replaced with X-operators, $\sigma^i_{N+i} = X_{N+i}$, and the adjustments κ^h_{aux} contain Z_{N+k} for every string p^k_{dat}, that anticommutes with h_{dat}. In [60] some concern is expressed that the operator weight might generally scale with the number of auxiliary qubits added - a key problem addressed by our work. We will in the following pick a set of strings $\{p^i_{\text{dat}}\}$ such that every term $h_{\text{dat}} \in \mathcal{S}$, resulting from any fermionic Hamiltonian, anticommutes with only a small number of stabilizers.

In Appendix 3.9.1 we give more details about these Auxiliary Qubit codes, such as their logical basis and the derivation of their stabilizers, adjustment terms as well as of the initialization unitaries $V_{\text{aux dat}}$. There are a few ways to extend the Auxiliary Qubit Mappings. In replacing the Pauli operators $\{\sigma^i_{N+i}\}$ with a set of Pauli strings $\{\gamma^i_{\text{aux}}\}$, we can even stabilize Pauli strings $\{p^i_{\text{dat}}\}$ that anticommute. In a similar vein, we can express the Verstraete-Cirac transform as a quantum code, which allows us to make modifications and to verify its operator transforms, see Appendix 3.9.3.

3.5 Auxiliary qubit mappings

3.5.1 E-type AQM

Here we present a mapping that remedies the biggest drawback of the S-pattern Jordan-Wigner transform under a moderate overhead of qubits. Given a $(\ell_1 \times \ell_2)$ block of data qubits, we are going to add ℓ_2 qubits as auxiliaries in computational basis. With this overhead, we will not manage to achieve any advantage for lattice models, but the scaling of long-range interactions (on the fermionic lattice) is improved. The following mapping will be referred to as E-type AQM. We will first illustrate its graph, along with instructions on how to initialize the stabilizer state from $|\varphi\rangle_{\text{dat}} \otimes |0^r\rangle_{\text{aux}}$. Afterwards, a discussion of the resulting Pauli strings will elucidate the advantages of the E-type AQM.

The idea of the E-type AQM is to store the parity of distinct data-qubit subsets permanently on auxiliary qubits. As we will see shortly, choosing to attach an auxiliary qubit to each of the ℓ_2 data-qubit rows is providing us with a geometric interpretation of the resulting strings. The result is shown in Figure 3.5(a). Note that two things are different between the S-pattern Jordan-Wigner transform and the E-type AQM: firstly, the connectivity graph has changed. A row of qubits is now coupled to one auxiliary qubit, and only those auxiliary qubits are coupled together, data

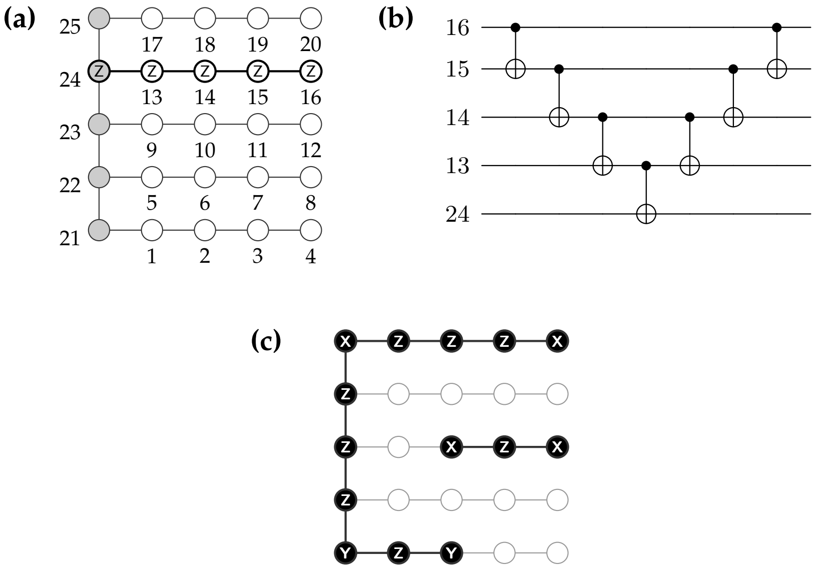


Figure 3.5. E-type AQM. **(a)** A block of (4×5) data qubits (white) enhanced with 5 auxiliary qubits (gray). A single stabilizer is highlighted in the graph. All qubits are labeled, where numberes 1-20 indicate the canonical ordering. **(b)** Initializing one of the stabilizers $(\bigotimes_{i=13}^{16} Z_i) \otimes Z_{24}$. **(c)** Simulating Pauli strings $\widetilde{h}_{\text{aux dat}}$ that are logical versions of $h_{\text{dat}} = (X \otimes Z \otimes \cdots \otimes Z \otimes X)$. The strings are highlighted as explained in Figure 3.2(b). While long strings are rerouted to skip rows, extending along the corresponding auxiliary qubits instead, shorter strings that do not switch rows can be simulated in parallel.

qubits in different rows are not coupled anymore. Although such connections between data qubits might be useful for simulating many-body terms, they are not necessarily required. Secondly, we have also changed the labeling of the qubits: the indices $i \in [\ell_1 \ell_2]$ still correspond to the indices attached to fermion operators in (2.11), but their order in the graph does no longer resemble an S-pattern of the canonical indices.

From $|\varphi\rangle_{\text{dat}} \otimes |0^r\rangle_{\text{aux}}$ the logical state $|\widetilde{\varphi}\rangle_{\text{aux dat}}$ can be initialized in $O(\ell_1)$-time and a total of $O(\ell_1 \ell_2)$ gates. Here a chain of CNOTs is used to mirror the collective parity information of an entire row of qubits on the attached auxiliary. The scaling in time is due to the fact that the preparation circuit in Figure 3.5(b), can theoretically be implemented on every row in parallel. The stabilizers of the system are

$$\left(\bigotimes_{i \,\in\, \text{row } k} Z_i \right) \otimes Z_{N+k} \,, \tag{3.8}$$

for all rows $k \in [\ell_2]$ in the data qubit block. We now turn to describe the resulting Pauli strings, for which we need to discuss the adjustments κ^h_{aux}. Diagonal terms (3.2) in the Hamiltonian do not influence the stabilizer state, as well as hopping terms (3.1) between qubits in the same row. Our attention is thus focused on Pauli strings of the form $h_{\text{dat}} = (X_i \otimes Z_{i+1} \otimes \cdots \otimes Z_{j-1} \otimes X_j)$, where qubits i and j are situated in different rows k and l, where $k < l$. Those Pauli strings are subsequently adjusted by $\kappa^h_{\text{aux}} = (X_{N+k} \otimes X_{N+l})$.

In order to make these terms more convenient, we multiply the adjusted strings with the corresponding stabilizers (3.8) of rows k', for all $k \leq k' < l$. Here we discover the benefit of this mapping: wherever Pauli strings act as Z-strings on entire rows, the parity is inferred instead from the auxiliary qubits attached. This limits the length of parity substrings and so Pauli strings (originating from hopping terms) have a maximal length $2\ell_1 + \ell_2$, instead of $\ell_1\ell_2$. This is not just a benefit in time and gates, but also allows us to simulate single-row strings at the same time as long strings spanning these rows, see Figure 3.5(c).
Although we expect the E-type AQM to be useful for problems long-range interactions, it has no advantage compared to the S-pattern Jordan-Wigner transform if one considers locally-interacting lattice Hamiltonians. With only single-row Pauli strings or strings between adjacent rows, no savings in gates and algorithmic depth can be anticipated. In the following, we will define a mapping that can transform those models into local qubit-Hamiltonians.

3.5.2 Square lattice AQM

Our main result, the square lattice AQM, is a mapping that requires a square lattice connectivity graph of $\ell_1 \times (2\ell_2 - 1)$ qubits for a $(\ell_1 \times \ell_2)$ fermionic lattice. With the large amount of $\ell_1(\ell_2 - 1)$ qubits added, we make sure that the code space can be initialized in $O(\ell_1)$ time steps; a time frame that is better than linear in the total number of data qubits. In the resulting mapping, we will be able to reroute and deform Pauli strings, such that strings originating from hopping terms have an operator weight of the order of the Manhattan distance between the two qubits on the lattice. The implication of this mapping for lattice Hamiltonians is that vertical hopping terms have a constant weight, and the algorithmic depth required to simulate such a model (after the stabilizer state is prepared) is constant, i.e. independent of the lattice dimension.

Before we start describing the mapping, we want to introduce some helpful notation concerning qubit labeling. For the sake of a geometric interpretation, we will migrate to a geometric labeling, where each qubit index denotes its coordinate on a grid. In the following, qubits in the data register will bear labels $(i,\, j) \in [\ell_1] \otimes [\ell_2]$, so each data qubit sits on integer positions of a grid and the qubit in the south-west corner of the block has coordinate $(1,\, 1)$. Beginning from that very qubit, the index of each qubit is given according to the canonical order of the S-pattern in Figure 3.4.

We will now describe the placement of the auxiliary qubits on the lattice. The idea of the square lattice AQM is to insert auxiliary qubits in between data qubits of different rows, so in between $(i,\, j)$ and $(i,\, j+1)$ into half-integer positions $(i,\, j+\frac{1}{2})$, in order to cancel the parity strings in between those qubits. However, we also want the p-strings to have (anti-)commutation relations like Majorana-pair operators. This is an integral ingredient to avoid long adjustments substrings κ^h_{aux}. To that end, we use a Hadamard-basis Auxiliary Qubit code with stabilizers

$$p_{\text{dat}}^{(i,j+\frac{1}{2})} \otimes X_{(i,j+\frac{1}{2})}\,, \tag{3.9}$$

which act on the data qubits at $(i,\, j)$ and $(i,\, j+1)$ as X- or Y-operators and as Z-operators on all other data qubits along the S-pattern in between them. The position of the auxiliary qubits and the choice of stabilizers can be seen in Figure 3.6. Note that it is unnecessary for the auxiliary qubits to be connected to each other in the horizontal direction, although it might come in handy in the process of initializing the code space. As indicated in the figure, the Pauli terms on $(i,\, j)$ and $(i,\, j+1)$ in the stabilizers of qubits $(i,\, j+\frac{1}{2})$ are different for even and odd rows numbers j. The sole reason for this decision is to render both terms of the vertical hopping terms with real coefficients (3.2) of the same weight. For every vertical connection $(i,\, j+\frac{1}{2})$, the p-substrings of the stabilizers

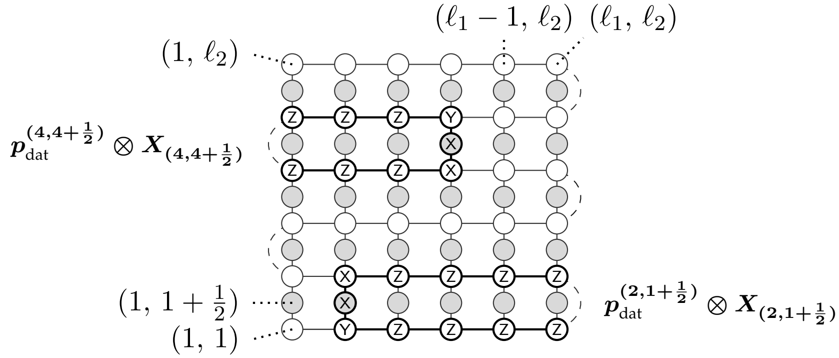

$(\ell_1-1,\,\ell_2)$
$(\ell_1,\,\ell_2)$
$(1,\,\ell_2)$
$\boldsymbol{p}_{\mathrm{dat}}^{(\mathbf{4},\mathbf{4}+\frac{1}{2})} \otimes \boldsymbol{X}_{(\mathbf{4},\mathbf{4}+\frac{1}{2})}$
Z
Z
Z
Y
X
Z
Z
Z
X
X
Z
Z
Z
Z
$(1,\,1+\frac{1}{2})$
X
$(1,\,1)$
Y
Z
Z
Z
Z
$\boldsymbol{p}_{\mathrm{dat}}^{(\mathbf{2},\mathbf{1}+\frac{1}{2})} \otimes \boldsymbol{X}_{(\mathbf{2},\mathbf{1}+\frac{1}{2})}$

Figure 3.6. Square lattice AQM, defined on a $\ell_1\times(2\ell_2-1)$ square lattice of qubits, here $\ell_1=\ell_2=6$. The gray qubits form the aux-register. Some qubits are labeled with their coordinates (dotted lines), where the auxiliary qubits generally sit on half-integer positions. The dashed lines do not couple qubits, but only indicate the windings of the S-pattern of the underlying Jordan-Wigner transform. The highlighted qubits and edges are two examples of stabilizers for odd and even rows, respectively, labeled in bold.

(3.9) are defined as:

$$p_{\mathrm{dat}}^{(i,\,j+\frac{1}{2})} = \left(\bigotimes_{k=i+1}^{\ell_1} Z_{(k,\,j)}\right)\left(\bigotimes_{l=\ell_1}^{i+1} Z_{(l,\,j+1)}\right) \otimes Y_{(i,\,j)} \otimes X_{(i,\,j+1)}\,, \quad \text{for odd } j, \tag{3.10}$$

$$= \left(\bigotimes_{k=i-1}^{1} Z_{(k,\,j)}\right)\left(\bigotimes_{l=1}^{i-1} Z_{(l,\,j+1)}\right) \otimes X_{(i,\,j)} \otimes Y_{(i,\,j+1)}\,, \quad \text{for even } j. \tag{3.11}$$

Now we are going to give instructions on how to initialize the state $|\widetilde{\varphi}\rangle$ within $O(\ell_1)$ depth, starting from a disentangled state $|\varphi\rangle_{\mathrm{dat}} \otimes |0^r\rangle_{\mathrm{aux}}$. First we apply Hadamard gates on all auxiliary qubits. In all rows with odd [even] row numbers j, we then simultaneously apply the strings $(Y_{(\ell_1,\,j)}\otimes X_{(\ell_1,\,j+1)})$ $\left[(X_{(1,\,j)} \otimes Y_{(1,\,j+1)})\right]$ conditional on the qubit at $(\ell_1,\, j+\frac{1}{2})$ $\left[(1,\, j+\frac{1}{2})\right]$. Entangling these auxiliaries is easy as the stabilizers are at the windings and therefore local, the operation can be performed in $O(1)$ time steps. We then proceed by applying the strings

$$X_{(\ell_1-s+1,\,j)} \otimes Y_{(\ell_1-s+1,\,j+1)} \otimes Y_{(\ell_1-s,\,j)} \otimes X_{(\ell_1-s+1,\,j+\frac{1}{2})} \otimes X_{(\ell_1-s,\,j+1)}$$
$$\left[Y_{(s,\,j)} \otimes X_{(s,\,j+1)} \otimes X_{(s+1,\,j)} \otimes X_{(s,\,j+\frac{1}{2})} \otimes Y_{(s+1,\,j+1)}\right] \tag{3.12}$$

conditionally on the qubits $(\ell_1 - s,\, j + \frac{1}{2})$ $[(s + 1,\, j + \frac{1}{2})]$. We do this sequentially from $s = 1$ to $s = (\ell_1 - 1)$, which means we require $O(\ell_1)$ time steps in total. This concludes the definition $V_{\text{aux dat}}$, as can be verified considering its formal definition in Appendix 3.9.1, and where we use that (3.12) is obtained from the multiplication of a p-string with the closest stabilizer. A measurement-based approach for state preparation is discussed in Section 3.7.

We are now going to describe the logical operators of the code space defined. In Figure 3.7(a), the adjusted term $\widetilde{h}_{\text{aux dat}}$ to a string $h_{\text{dat}} = (X \otimes Z \otimes \cdots \otimes Z \otimes X)$ is presented. In Section 3.9.3.1 we will show that for Pauli strings originating from hopping terms (3.2) between two sites (i, j) and $(k,\, l)$, it is sufficient to check for adjustments on only the auxiliary qubits at $(i,\, j \pm \frac{1}{2})$ and $(k,\, l \pm \frac{1}{2})$. If j and l are different rows, it follows that the string is not continuous, see Figure 3.7(a). We then choose to multiply the adjusted term with the stabilizers involving the auxiliary qubits on which we wish the string to cross rows. For vertical hoppings of lattice Hamiltonians, this choice is trivial. For arbitrary hoppings however it is not. Considering that we likely have several such terms inside one Hamiltonian, we want commuting strings not to overlap so we would deform them (by multiplying other stabilizers) to go around each other. This allows us to simulate them in parallel. In Figure 3.7, panels (b)-(d), different paths have been chosen for the logical operator $\widetilde{h}_{\text{aux dat}}$ to run along. Only deformed by the multiplication of stabilizers, all of those choices are in fact equivalent. Note that taking a direct path, the resulting strings will always be of roughly the same length, as every direct path connecting two nodes on a square lattice has the same distance: the Manhattan distance.

In the following, we will generalize this mapping to yield an AQM-version that requires fewer auxiliary qubits.

3.5.3 Sparse AQM

The sparse AQM is a modification of the square lattice AQM that allows us to make a trade-off between the number of auxiliary qubits required and the locality in the resulting strings. The latter directly influences the performance of any quantum simulation algorithm.

In the square lattice AQM, each data qubit (of the interior) has two nonlocal connections in the vertical direction. This can be regarded as

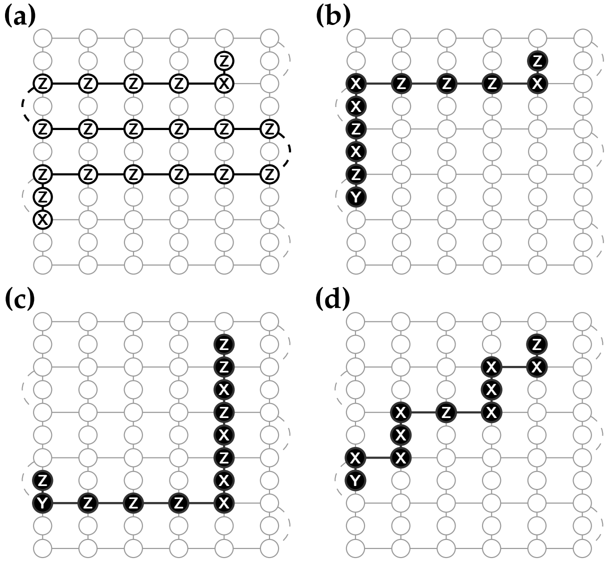


Figure 3.7. Depicted are logical representatives of the same hopping term $h_{\text{dat}} = (X \otimes Z \otimes \cdots \otimes Z \otimes X)$ spanning several rows and columns in the square lattice. The depiction of all strings follows the explanation in Figure 3.2(b). **(a)** Adjusted term $(h_{\text{dat}} \otimes \kappa^{h}_{\text{aux}})$, not yet multiplied with any stabilizer. Note that this string is not connected on the lattice, and the windings on which the string is disconnected are highlighted. **(b)-(d)** Pauli strings $\widetilde{h}_{\text{aux}\,\text{dat}}$ that are equivalent to $(h_{\text{dat}} \otimes \kappa^{h}_{\text{aux}})$ by multiplication with stabilizers. All those strings are continuous on the connectivity graph. The strings in (b) and (d) have the same weight (and the string in (c) is just slightly longer) which is determined by the Manhattan distance of the string endpoints.

quite wasteful, as a mapping with fewer vertical connections would work in the same way while effectively reducing the number of auxiliary qubits. Here we introduce the sparse AQM, in which vertical connections have a certain distance from each other. Let us say vertical connections are always placed $\mathcal{I}$ qubits apart. The periodicity $\mathcal{I}$ thus becomes a parameter of the mapping and is generally an integer number $\mathcal{I} \in [\ell_1 - 1]$, where the case $\mathcal{I} = 1$ reproduces the square lattice AQM. We have excluded the case in which we have only one vertical connection between every pair of rows, as it is covered by the E-type AQM already. For convenience let us say that $(\ell_1 - 1)/\mathcal{I}$ is an integer such that we can place vertical connections at the right and left boundary of the grid without spacing unequally. The connectivity graph that puts auxiliary qubits on half integer positions along $\mathcal{I}$-spaced columns can be seen in Figure 3.8(a), along with the typical stabilizers. In this mapping the auxiliary register holds $r = (\ell_2 - 1) \cdot (\frac{\ell_1 - 1}{\mathcal{I}} + 1)$ qubits, which is somewhere in between the square lattice and E-type AQM. For the initialization circuit, $V_{\text{aux dat}}$, the sequence (3.12) has to be changed into applying the strings

$$\begin{gathered}\left(X_{(\ell_1 - s + \mathcal{I},\, j+\frac{1}{2})} \otimes p_{\text{dat}}^{(\ell_1 - s + \mathcal{I},\, j+\frac{1}{2})}\right) \cdot p_{\text{dat}}^{(\ell_1 - s,\, j+\frac{1}{2})} \\ \left[\left(X_{(s+1-\mathcal{I},\, j+\frac{1}{2})} \otimes p_{\text{dat}}^{(s+1-\mathcal{I},\, j+\frac{1}{2})}\right) \cdot p_{\text{dat}}^{(s+1,\, j+\frac{1}{2})}\right]\end{gathered} \tag{3.13}$$

conditionally on qubits $(\ell_1 - s,\, j + \frac{1}{2})$ $[(s+1,\, j+\frac{1}{2})]$
for $s = \mathcal{I},\, 2\mathcal{I},\, 3\mathcal{I},\, \dots,\, \ell_1 - 1$. All those strings in the sequence are of weight $O(\mathcal{I})$, but there are just $(\ell_1 - 1)/\mathcal{I}$ of them, which brings the depth of the entire circuit to $O(\ell_1)$.

Figure 3.8(b) shows some output strings of this mapping. While crossing rows works like in the square lattice AQM, the sparsity of vertical connections makes for a more limited choice on where the strings can run along. As a consequence, hopping terms between modes with a horizontal distance smaller than $\mathcal{I}$ will transform into strings like in the E-type mapping. The effect of sparsity on simulations of a lattice model is discussed in the following section.

Note that we have made two arbitrary design choices for the connectivity graph of this mapping: firstly, we have chosen for the auxiliary qubits to be situated in between rows of data qubits. In order to fit this mapping to a compact square lattice, we can take the auxiliary qubits from in between the rows and insert them into the rows, so e.g. take them from $(i,\, j+\frac{1}{2})$ and insert them at $(i+\frac{1}{2},\, j)$. Then, the auxiliaries have to be connected to the data qubits $(i,\, j)$ and $(i+1,\, j)$, as well as the auxiliary qubits at $(i+\frac{1}{2},\, j\pm 1)$. In the end, no qubits will be in the spaces between rows - this makes the array more dense and we can map it to a square lattice, but also requires us to skip auxiliary qubits in some horizontal hopping strings. Secondly, we have decided to place auxiliary qubits inside the same column of every other vertical connection. Alternatively, the vertical connections could be arranged in a brickwork pattern in order to minimize the weight of the adjustments κ^{h}_{aux}, but then vertical connections along a straight line are no longer possible.

3.6 Example: Fermi-Hubbard lattice model

3.6.1 Second quantization and Jordan-Wigner transform

Here we demonstrate the use of AQMs on the Fermi-Hubbard model. In this model, we describe spin-$\frac{1}{2}$ fermions hopping on a square lattice, with a repulsion term whenever spin-up and -down particles are present on the same site. In the following, we will describe the Hamiltonian in both, second quantization and in terms of Pauli strings after Jordan-Wigner transform. Investigating the shortcomings of this mapping with respect to circuit depth will be the motivation for the application of AQMs in the next step. Let us consider an $(L\times L)$-site square lattice of spatial sites populated by spin-(1/2) fermions: as every such site hosts a spin-up and -down mode, a total of $N=2L^2$ qubits are minimally required. For convenience, the spin-up and -down modes of the fermionic site with the physical location $(x,\, y)$ shall be placed at the coordinates $(2x,\, y)$ and $(2x-1,\, y)$ in the two-dimensional embedding. This means the spin-partners are horizontal neighbors, which is advantageous for the Jordan-Wigner transform (and square lattice AQM). The Fermi-Hubbard Hamil-

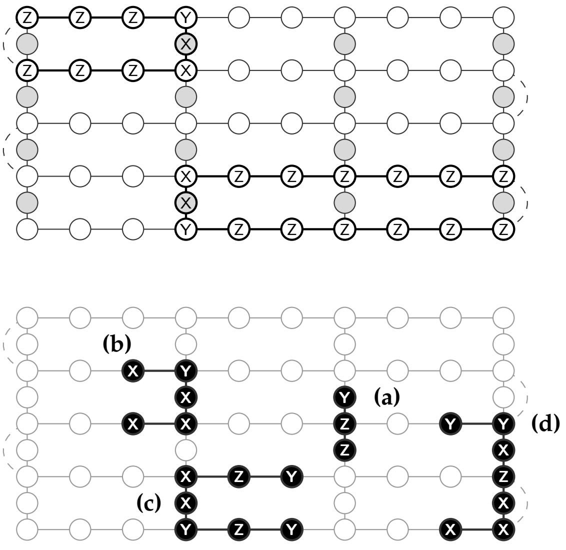


Figure 3.8. Sparse AQM with a periodicity of three ($\mathcal{I} = 3$). **Top:** Structure and stabilizers. The gray qubits are auxiliaries, placed sparsely on half-integer positions, connecting different rows. We depict one of the stabilizers in an odd and an even row, respectively. **Bottom:** Logical equivalents $\widetilde{h}_{\text{aux dat}}$ of various strings $h_{\text{dat}} = (X \otimes Z \otimes \cdots \otimes Z \otimes X)$, that originate from vertical hopping terms. **(a)** A vertical hopping along a vertical connection. The mapping yields the same $(Z \otimes Z \otimes Y)$-string as we would expect from the square lattice AQM. **(b)** The string is connecting $(3,\,3)$ and $(3,\,4)$. This example shows the virtue of the sparse AQM: the parity string takes a shortcut along the closest vertical connection. **(c)** Here we connect the qubits on $(6,\,1)$ and $(6,\,2)$ from the other direction: over the vertical connection between $(4,\,1)$ and $(4,\,2)$. **(d)** A next-nearest-neighbor vertical hopping term between $(9,\,1)$ and $(9,\,3)$.

tonian is defined as

$$\overbrace{\sum_{(i,j)} \left(t_{ij}^{\leftrightarrow}\; c^{\dagger}_{(i,\,j)} c_{(i+2,\,j)} + \text{h.c.} \right)}^{\text{horizontal hoppings}} + \overbrace{\sum_{(i,\,j)} \left(t_{ij}^{\updownarrow}\; c^{\dagger}_{(i,\,j)} c_{(i,\,j+1)} + \text{h.c.} \right)}^{\text{vertical hoppings}}$$

$$+ \underbrace{\sum_{(i,\,j)} \epsilon_{ij}\; c^{\dagger}_{(i,\,j)} c_{(i,\,j)}}_{\text{on-site detunings}} + \underbrace{\sum_{(2i,\,j)} U_{ij}\; c^{\dagger}_{(2i,\,j)} c_{(2i,\,j)} c^{\dagger}_{(2i-1,\,j)} c_{(2i-1,\,j)}}_{\text{Hubbard interactions}}\,, \qquad (3.14)$$

where $t_{ij}^{\leftrightarrow}$, $t_{ij}^{\updownarrow}$, ϵ_{ij} and U_{ij} are real parameters. In this particular example sums run over all possible coordinates $(i,\, j)$, $(2i,\, j)$ respectively, but implement open boundary conditions. With an S-pattern Jordan-Wigner transform, the Hamiltonian can now be mapped onto an $(2L \times L)$ square lattice of qubits:

$$\begin{aligned} H = &\sum_{(i,\,j)} \frac{t_{ij}^{\leftrightarrow}}{2} \left(X_{(i,\,j)} \otimes Z_{(i+1,\,j)} \otimes X_{(i+2,\,j)} + Y_{(i,\,j)} \otimes Z_{(i+1,\,j)} \otimes Y_{(i+2,\,j)} \right) \\ &+ \sum_{(i,\,j),\,\mathrm{odd}\,j} \frac{t_{ij}^{\updownarrow}}{2} \left(\bigotimes_{k=i+1}^{2L} Z_{(k,\,j)} \right) \left(\bigotimes_{l=2L}^{i+1} Z_{(l,\,j+1)} \right) \left(X_{(i,\,j)} \otimes X_{(i,\,j+1)} + Y_{(i,\,j)} \otimes Y_{(i,\,j+1)} \right) \\ &+ \sum_{(i,\,j),\,\mathrm{even}\,j} \frac{t_{ij}^{\updownarrow}}{2} \left(\bigotimes_{k=i-1}^{1} Z_{(k,\,j)} \right) \left(\bigotimes_{l=1}^{i-1} Z_{(l,\,j+1)} \right) \left(X_{(i,\,j)} \otimes X_{(i,\,j+1)} + Y_{(i,\,j)} \otimes Y_{(i,\,j+1)} \right) \\ &+ \sum_{(i,\,j)} \frac{\epsilon_{ij}}{2} \left(\mathbb{I} - Z_{(i,\,j)} \right) + \sum_{(2i,\,j)} \frac{U_{ij}}{4} \left(\mathbb{I} - Z_{(2i,\,j)} \right) \left(\mathbb{I} - Z_{(2i-1,\,j)} \right) . \end{aligned} \tag{3.15}$$

Let us discuss the terms of this Hamiltonian, and finally arrive at the shortcomings of the mapping applied. We note that the vertical hopping terms are different with respect to even and odd columns, due to different directions of the S-pattern. All terms but the vertical hoppings have a constant weight and can be simulated in $O(1)$ time: only the latter can assume a length of up to $4L$. Unfortunately, we have $O(L)$ terms of weight $O(L)$ per row pair. Although these strings commute, they do overlap, which means we cannot simulate them in parallel: if no cancellations are possible, then the entire algorithm has an algorithmic depth of $O(L^2)$, so it scales with the lattice area. In this case the simulation time and the gate count cannot be better than being proportional to the total number of qubits, which renders increasing lattice size expensive. If the simulation algorithm allows us to cancel substrings of consecutively simulated Pauli strings (see for instance [39]), the algorithmic depth can improve to up to $O(L)$. To achieve even better scalings, we will employ the square lattice AQM and sparse AQM on (3.14). A detailed consideration of the E-type AQM is omitted, as it does not improve upon the scaling in case of lattice models.

3.6.2 Square lattice and sparse AQM

With the square lattice AQM, the Fermi-Hubbard Hamiltonian can be simulated in constant time, neglecting the algorithmic depth necessary to initialize the code space, which is $O(L)$ or $O(1)$ depending on the exact method used. We will now describe how the square lattice AQM modifies the terms of the Hamiltonian (3.15), after which we will discuss the sparse AQM in that regard.

We now use the square lattice AQM to render the vertical hopping terms local: after adjusting each term of (3.15) by $h_{\text{dat}} \rightarrow h_{\text{dat}} \otimes \kappa^{h}_{\text{aux}}$, the multiplication of adjusted hopping terms between $(i,\, j)$ and $(i,\, j+1)$ with stabilizers $(p_{\text{dat}}^{(i,\, j+\frac{1}{2})} \otimes X_{(i,\, j+\frac{1}{2})})$ is resulting in local operators of weight 3. While the hopping terms in (3.14) only have real coefficients, the operator weight of more general vertical hopping terms varies, but remains 3 on average. For complex hopping amplitudes $t^{\updownarrow}_{ij}$, we find

$$\begin{aligned} t^{\updownarrow}_{ij}\; c^{\dagger}_{(i,\, j)} c_{(i,\, j+1)} + (t^{\updownarrow}_{ij})^{*}\; c^{\dagger}_{(i,\, j+1)} c_{(i,\, j)} \quad &\hat{=} \\ \frac{(-1)^{j}}{2} \operatorname{Re}(t^{\updownarrow}_{ij})\; &\left(Z_{(i,\, j-\frac{1}{2})} \otimes Z_{(i,\, j)} \otimes Y_{(i,\, j+\frac{1}{2})} \right) \\ - \frac{(-1)^{j}}{2} \operatorname{Re}(t^{\updownarrow}_{ij})\; &\left(Y_{(i,\, j+\frac{1}{2})} \otimes Z_{(i,\, j+1)} \otimes Z_{(i,\, j+\frac{3}{2})} \right) \\ + \frac{(-1)^{j}}{2} \operatorname{Im}(t^{\updownarrow}_{ij})\; &\left(Z_{(i,\, j-\frac{1}{2})} \otimes Z_{(i,\, j)} \otimes X_{(i,\, j+\frac{1}{2})} \otimes Z_{(i,\, j+1)} \otimes Z_{(i,\, j+\frac{3}{2})} \right) \\ - \frac{(-1)^{j}}{2} \operatorname{Im}(t^{\updownarrow}_{ij})\;\; &X_{(i,\, j+\frac{1}{2})} \,. \end{aligned} \tag{3.16}$$

The improvements that we make on vertical terms come at the cost of the adjustments κ^{h}_{aux} to other terms in (3.15). However, as already mentioned, the structure of the strings $\{p^{i}_{\text{dat}}\}$ guarantees to keep those other terms local. For horizontal hopping terms that are (like the vertical strings) of the form $h_{\text{dat}} = (\mathrm{A}_i \otimes Z_{i+1} \otimes ... \otimes Z_{j-1} \otimes \mathrm{B}_j)$, with $\mathrm{A}, \mathrm{B} \in \{X,\, Y\}$, the substrings κ^{h}_{aux} invoke Z-operators at the end of the strings which makes for an additional weight of 2. On the other hand, if $\mathrm{A}, \mathrm{B} = Z$, κ^{h}_{aux} features Z-operators along the entire string. This means that while single Z-operators are in this way adjusted to $Z_{(i,\, j)} \mapsto Z_{(i,\, j-\frac{1}{2})} \otimes Z_{(i,\, j)} \otimes Z_{(i,\, j+\frac{1}{2})}$, the two-qubit Hubbard terms gain 4 qubits worth of weight.

With the square lattice AQM, we have thus managed to reduce the weight of every term to a constant independent of the system size. A list of relevant terms, that compares Jordan-Wigner and square lattice AQM can be found in Tables 3.2 and 3.3. Having achieved locality of every Hamiltonian term, we can trotterize $\widetilde{H}_{\text{aux}\,\text{dat}}$ by for instance applying all horizontal hopping terms in $O(1)$ time, then continue with a time slice in which we simulate all vertical hoppings, follow-up with all on-site interactions and Hubbard terms, and so on. Alternatively, one may apply Hamiltonian simulation strategies to simulate patches of the lattice more accurately and then interweave these patches with the HHKL algorithm, [68].

With the square lattice AQM, we have made the simulation scalable in terms of algorithmic depth and gate count. The requirement on the qubit number has however almost doubled. In order to be more economic with the number of auxiliary qubits, we consider the sparse AQM, which will help us to maximize the size of the simulated lattice on a fixed qubit budget. Placing vertical connections $\mathcal{I}$ qubits apart, the required number of auxiliary qubits is $r = \left(\frac{2L^2-2L+1}{\mathcal{I}} + L - 1\right)$. The weight of vertical hopping strings now largely depend upon their distance to the next vertical connection: let us say there is a vertical connection across $(i,\, j + \frac{1}{2})$, then the vertical hoppings between $(i,\, j)$ and $(i,\, j+1)$ are of (constant) weight 3, like in the square lattice AQM, while the vertical hoppings of modes to their left and right rather resemble the strings of E-type AQM. The worst case is certainly met for vertical hoppings in the middle of two vertical connections, so between $(i \pm \frac{1}{2}\mathcal{I},\, j)$ and $(i \pm \frac{1}{2}\mathcal{I},\, j + 1)$. Thus per vertical connection, there are $O(\mathcal{I})$ strings of weight $O(\mathcal{I})$ overlapping with one another. The simulation time is thus $O(\mathcal{I})$ if we allow cancellations and $O(\mathcal{I}^2)$ in the general case.

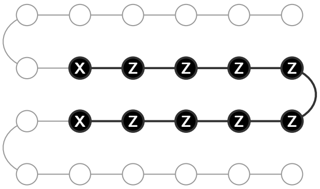
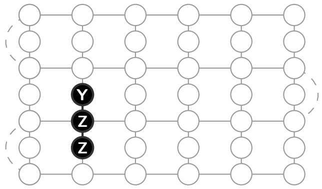
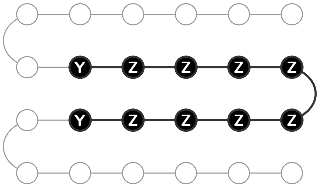
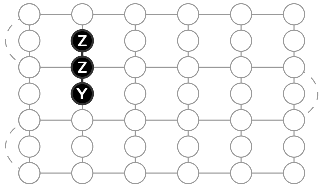
Jordan-Wigner transform		Square lattice AQM
		
$\left(\bigotimes_{k=i+1}^{2L} Z_{(k,j)}\right)\left(\bigotimes_{l=2L}^{i+1} Z_{(l,j)}\right) \otimes X_{(i,j)} \otimes X_{(i,j+1)}$	$\mapsto$	$-\left(Z_{(i,j-\frac{1}{2})} \otimes Z_{(i,j)} \otimes Y_{(i,j+\frac{1}{2})}\right)$
		
$\left(\bigotimes_{k=i+1}^{2L} Z_{(k,j)}\right)\left(\bigotimes_{l=2L}^{i+1} Z_{(l,j)}\right) \otimes Y_{(i,j)} \otimes Y_{(i,j+1)}$	$\mapsto$	$Y_{(i,j+\frac{1}{2})} \otimes Z_{(i,j+1)} \otimes Z_{(i,j+\frac{3}{2})}$

Table 3.2. Comparing the Jordan-Wigner transform (3.15) to square lattice AQM when applied to the Hubbard model (3.15). In this table we present vertical hopping terms – the strings of which the nonlocal part is canceled. We generally compare Jordan-Wigner strings, h_{dat} (left), to their logical equivalents $\widetilde{h}_{\text{aux dat}}$ (right) in the AQM. The strings are depicted geometrically (following the explanation of Figure 3.2(b)) and symbolically (below the drawings). Note that we display hoppings between odd rows j and even rows $j+1$ only. For j even, the two $\widetilde{h}_{\text{aux dat}}$-terms are exchanged.

3.6.3 VCT and BKSF

The Fermi-Hubbard model can also be made local by the Verstraete-Cirac transform or Superfast simulation. In this section, we will compare the weights of Pauli strings appearing in those cases to the strings resulting from transforming the Hubbard model with the square lattice AQM. We have compiled a list of the operator weights in Table 3.4, and the interested reader may find a visual representation of the strings from

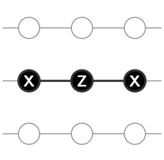
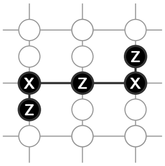
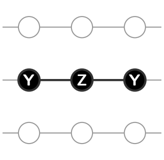
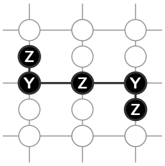
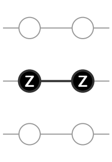
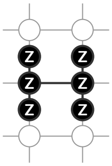
Jordan-Wigner transform		**Square lattice AQM**
		
$X_{(i,j)} \otimes Z_{(i+1,\,j)} \otimes X_{(i+2,\,j)}$	$\mapsto$	$Z_{(i,\,j-\frac{1}{2})} \otimes X_{(i,\,j)} \otimes Z_{(i+1,\,j)}$ $\otimes\ X_{(i+2,\,j)} \otimes Z_{(i+2,\,j+\frac{1}{2})}$
		
$Y_{(i,j)} \otimes Z_{(i+1,\,j)} \otimes Y_{(i+2,\,j)}$	$\mapsto$	$Z_{(i+2,\,j-\frac{1}{2})} \otimes X_{(i,\,j)} \otimes Z_{(i+1,\,j)}$ $\otimes\ X_{(i+2,\,j)} \otimes Z_{(i,\,j+\frac{1}{2})}$
		
$Z_{(i,j)} \otimes Z_{(i+1\,j)}$	$\mapsto$	$\bigotimes_{k\in\{0,1\}} \Big(Z_{(i+k,\,j-\frac{1}{2})}$ $\otimes Z_{(i+k,j)} \otimes Z_{(i+k,\,j+\frac{1}{2})}\Big)$

Table 3.3. Comparing the Jordan-Wigner transform (3.15) to square lattice AQM when applied to the Hubbard model (3.15). In this table, we present the horizontal hopping and Hubbard terms – all the Pauli strings that gain in weight. However, the addition in weight is constant and local, as shown in 3.9.3. We generally compare Jordan-Wigner strings, h_{dat} (left), to their logical equivalents $\widetilde{h}_{\text{aux dat}}$ (right) in the AQM. The strings are depicted geometrically (following the explanation of Figure 3.2(b)) and symbolically (below the drawings). Not on display are the on-site terms and single-qubit contributions from Hubbard interactions, $Z_{(i,\,j)}$, which are adjusted into $(Z_{(i,\,j-\frac{1}{2})} \otimes Z_{(i,\,j)} \otimes Z_{(i,\,j+\frac{1}{2})})$.

BKSF and VCT in Appendix 3.9.3. Let us briefly discuss how the weights of the terms come to be. The VCT and AQM are quite similar in the sense that both concatenate the Jordan-Wigner transform with a quantum code. However, the data-qubit substrings of the VCT stabilizers just consist of Z-strings, which has two consequences: firstly, the stabilizers commute with diagonal terms like on-site detunings and Hubbard interactions, leaving them unadjusted and without any gain of weight. With this feature, the VCT distinguished itself from the other mapping in producing strings of the lowest weight. Secondly, while in the AQM a hopping string would just be adjusted on its end points, adjustments have to be made all along the strings in the VCT: fortunately, the auxiliary-qubit substring of the VCT stabilizers cancel these adjustments, causing this mapping to have shorter strings in the vertical direction (see Section 3.7). We thus place spin-up and -down modes of the same spatial site vertically adjacent, like we have placed them horizontally adjacent in the AQM. This leads to the weights of horizontal and vertical hoppings to be interchanged between VCT and AQM (on average). The stabilizers of both mappings can be made local with a weight of 6 (and weight-3 stabilizers at the boundaries), which is also the weight of stabilizers in the BKSF. The BKSF, defined on the least amount of qubits, has surprisingly the longest strings. The reason for this is that logical Z-operators have weight 4 - a consequence of the square lattice connectivity. With this, the BKSF has also the largest variety of weights in hopping strings, while in the VCT, there is no variety at all among strings in the same direction. While the VCT appears to be the favorable option when comparing string lengths (followed by the AQM), it also uses the most qubits, as becomes apparent in Appendix 3.9.3.

3.7 Comparison of AQM, VCT and BKSF

In this section, we will compare the Auxiliary Qubit Mapping, Superfast simulation and Verstraete-Cirac transform. Not only can the latter two be used to simulate the Hubbard model with local interactions, but we can also give them the Manhattan-distance property to align them with our notions of a good mapping for square lattices of qubits. This is done in Appendix 3.9.3. The reader completely unfamiliar with those mappings may also find an introduction reviewing the original proposals [25, 26]. Let us here compare AQM, VCT and BKSF regarding state

	Square lattice		
	AQM	VCT	BKSF
Stabilizer (interior)	6	6	6
Vertical hoppings $XX\,|\,YY\,|\,XY\,|\,YX$	$3|3|5|1$	$5|5|5|5$	$2|6|5|4$
Horizontal hoppings $XX\,|\,YY\,|\,XY\,|\,YX$	$5|5|5|5$	$3|3|3|3$	$8|4|5|7$
Two-qubit Hubbard terms	6	2	$6+2$
On-site terms	3	1	4

Table 3.4. String lengths of the Fermi-Hubbard model transformed by all three mappings. We compare the weight of the Pauli strings, that originate from the square lattice AQM, the Verstraete-Cirac transform and the Superfast simulation. For hopping terms, we consider the strings $h_{\mathrm{dat}} = (\mathrm{A}_i \otimes Z_{i+1} \otimes \cdots \otimes Z_{j-1} \otimes \mathrm{B}_j)$, with all variations of $\mathrm{A}, \mathrm{B} \in \{X,\, Y\}$. For vertical hoppings (in the AQM) we fix the case of j being in an even row. Two-qubit Hubbard terms are of the form $h_{\mathrm{dat}} = (Z \otimes Z)$, and on-site terms are singular Z-operators. In the BKSF it is required to skip a qubit, which we penalize with an additional cost of two gates. In conclusion, the Verstraete-Cirac transform seems to exhibit the shortest strings, with the weights of the hopping terms being the same for all $\mathrm{A}_i,\, \mathrm{B}_j$. Regarding string lengths, the square lattice AQM is in between the Verstraete-Cirac transform and the Superfast simulation, where the latter has the longest strings and largest variations in length.

preparation, qubit requirements, Manhattan-distance property and the possibility of error mitigation. Afterwards, we can conclude and identify cases in which each mapping is advantageous.

State preparation - As we have shown, there is a unitary quantum circuit for the AQM to elevate an N-qubit state to its equivalent in the logical basis. The VCT on the other hand has a logical basis that is entangled in a more complicated way, such that we cannot find a unitary quantum circuit of the same simplicity. Although the BKSF has no clear distinction between data and auxiliary qubits, there is a set of $N-1$ qubits that is only relevant for an S-pattern and one could argue that only vertical connections add the remaining qubits and introduce stabilizers. As each connection is implemented by just one entangled qubit, we believe that there might be a unitary circuit as simple as $V_{\text{aux dat}}$. As of now, we would have to resort to syndrome measurements to initialize the code space of VCT and BKSF. By syndrome measurements, we mean the measurement and readout of a generating set of stabilizers and correct for outcomes inconsistent with the code space. While measurement and readout-times of state-of-the-art quantum devices might make this strategy challenging at present, we can at least arrange for local stabilizers such that the time overhead per measurement cycle is constant. In Figure 3.9(c)-(d) the local stabilizer tilings of VCT and BKSF are shown. A planar tiling for stabilizers of square lattice and sparse AQM follows from multiplication of adjacent stabilizer generators

$$\left(p_{\text{dat}}^{(i,j+\frac{1}{2})} \otimes X_{(i,\,j+\frac{1}{2})}\right) \cdot \left(p_{\text{dat}}^{(i+1,\,j+\frac{1}{2})} \otimes X_{(i+1,\,j+\frac{1}{2})}\right) \quad \text{and}$$
$$\left(p_{\text{dat}}^{(i,\,j+\frac{1}{2})} \otimes X_{(i,\,j+\frac{1}{2})}\right) \cdot \left(p_{\text{dat}}^{(i+\mathcal{I},\,j+\frac{1}{2})} \otimes X_{(i+\mathcal{I},\,j+\frac{1}{2})}\right) \,, \tag{3.17}$$

excluding the stabilizers at the windings, which are local already. The result is a repeating pattern of tiles with *ears* at the windings, shown in Figure 3.9(a)-(b). Note that we have implicitly used these tilings already in the respective definitions of $V_{\text{aux dat}}$. While with the unitary quantum circuit we can prepare the state on only the data qubits before encoding it into the logical basis, the same thing seems impossible with syndrome measurements. Even if the protective operations would not change the data-qubit state, there is still an ambiguity in the logical bases of VCT and AQM, that we now want to discuss. As can be seen in Appendix 3.9.1, the quantum code layer included in these mappings transform any computational basis state $|\boldsymbol{\omega}\rangle_{\text{dat}}$ into a logical basis state $\left[\prod_{i\in[r]} \frac{1}{\sqrt{2}}(\mathbb{I} + S^i_{\text{aux dat}})\right] |\boldsymbol{\omega}\rangle_{\text{dat}} \otimes |\boldsymbol{\chi}\rangle_{\text{aux}}$, where $\{S^i_{\text{aux dat}}\}_i$ is a generating set of stabilizers and $\boldsymbol{\chi} = (\chi_1, \chi_2, \ldots, \chi_r)^\top \in \mathbb{Z}_2^{\otimes r}$ is a constant binary vector. While in the VCT, the set of stabilizers limit (not constrain) the choice of $\boldsymbol{\chi}$, (square lattice and sparse) AQMs are properly stabilized for all possible $\boldsymbol{\chi} \in \mathbb{Z}_2^{\otimes r}$. However, for both mappings the (signs of) adjustments made to operators h_{dat} depend on $\boldsymbol{\chi}$. For AQMs we rely on $\boldsymbol{\chi} = (0)^{\otimes r}$ for the substrings κ^h_{aux} to be free of signs. Obviously, for any basis with an unintended $\boldsymbol{\chi}$-shift, the logical Hamiltonian $\widetilde{H}_{\text{aux dat}}$ will not replicate the action of H_{dat}. As we cannot detect this $\boldsymbol{\chi}$-offset, we have to ignore it, e.g. pretend that $|\boldsymbol{\chi}\rangle = |0^r\rangle$ in AQMs: this effectively means that the state $|\widetilde{\varphi}\rangle_{\text{aux dat}}$, which is created with an unknown $\boldsymbol{\chi}$-shift in the aux-register, becomes a state $[\prod_i (p^i_{\text{dat}})^{\chi_i}]\, |\widetilde{\varphi}\rangle_{\text{aux dat}}$ without shift, a state we have not intended to prepare. To combat ambiguities in all mappings, the system has to be constrained to the correct subspace before any state preparation can happen. This means we have to measure not only the stabilizers, but also logical operators until all degrees of freedom are eliminated. Apart form the tiles, we could measure all logical Z-operators, i.e. all logical encodings of $(2c^\dagger_j c_j - 1)$. When all measurement outcomes yield '+1', we have prepared the logical zero state, $\widetilde{|0^N\rangle}_{\text{aux dat}}$. From there on, we directly prepare $|\widetilde{\varphi}\rangle_{\text{aux dat}}$ by e.g. Givens rotations [30, 69] using logical operators. This strategy appears to be the only option for measurement-based preparation of states in any mapping, although practically one will certainly want to perform only one cycle of measurements form the outcome of which the logical state and the (signs of the) stabilizers are defined. For the modest E-type AQM on the other hand, neither syndrome measurements nor unitary quantum circuits are necessary to prepare a logical state. Due to the fact that its logical basis is in the computational basis, the product state $(|0^N\rangle_{\text{dat}} \otimes |0^r\rangle_{\text{aux}})$ is in fact the logical zero state, even though the two registers are obviously not entangled. Initializing all qubits in zero at first is thus a sufficient preliminary to prepare the state $|\widetilde{\varphi}\rangle_{\text{aux dat}}$ with logical operators.

Qubit requirements - For all mappings we find the highest number of qubits they require to be $\leq 2N$, in fact only the VCT demands exactly

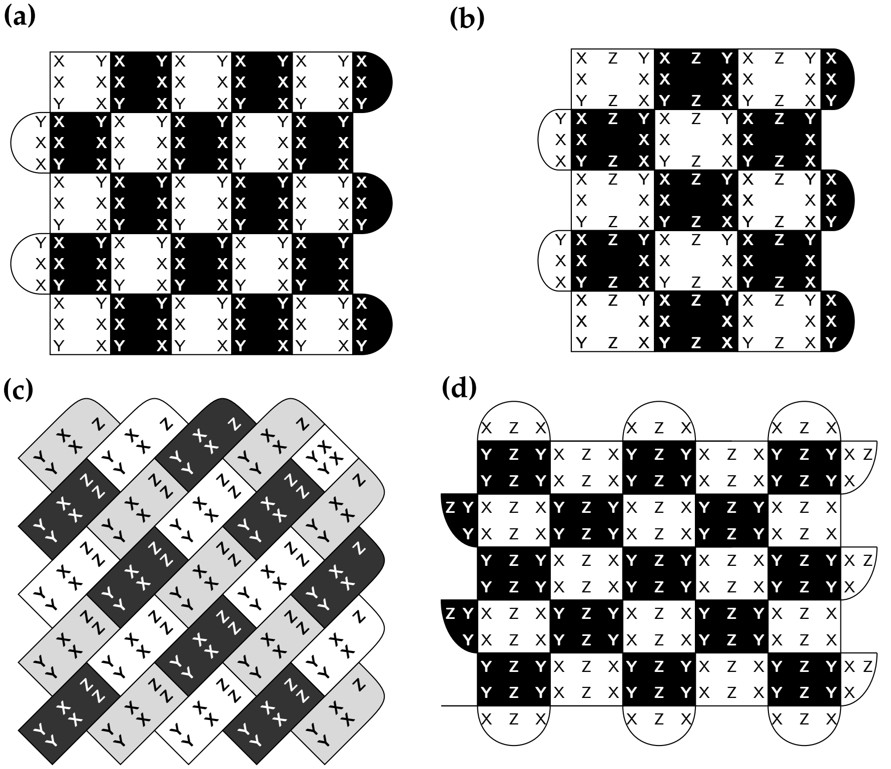


Figure 3.9. Tilings of local stabilizers for square lattice and sparse AQMs, BKSF and VCT. Every tile represents a local stabilizer involving qubits along its perimeter. Inside the tiles, X, Y and Z indicate the Pauli operators that every qubit contributes to the corresponding stabilizer. We have shaded the tiles to as a visual aid for error mitigation. **(a)** Square lattice AQM with dimensions $\ell_1 = \ell_2 = 6$. The stabilizers of all tiles are the same, except at the windings. **(b)** Sparse AQM with dimensions $\ell_1 = 7$, $\ell_2 = 6$ and $\mathcal{I} = 2$. **(c)** BKSF of a $\ell_1 = \ell_2 = 6$ fermionic lattice. The tiling is a three-colorable brickwork pattern. **(d)** VCT with dimensions $\ell_1 = \ell_2 = 6$. The stabilizer tiles are alternating in a checkerboard pattern, that resembles the rotated surface code except for the Z-operators on the data qubits.

$2N$ qubits, the square lattice AQM on the other hand requires ℓ_1 qubits less, and the BKSF requires even ℓ_2 less than the AQM. As for the AQM, we can think about reducing the amount of qubits with sparse AQMs. For the VCT such a modification is discussed in Appendix 3.9.3. As the qubits added to the VCT are generally added into the rows, its sparse version can be mapped back to a compact square lattice more easily than the AQM. In the BKSF, we can also make vertical connections more sparse, but as its layout is rotated, mapping the sparse BKSF to a compact square lattice requires changes in the connectivity graph, which will influence the continuity of resulting strings.

Manhattan-distance property - With all mappings we manage to transform long-range hopping terms of a $\ell_1 \times \ell_2$ fermionic lattice to continuous Pauli strings on a qubit lattice, that can be deformed by the multiplication of stabilizers. For all mappings, the shortest version of those strings involve a number of qubits scaling with the Manhattan distance of the fermionic modes on their lattice, but their exact weight differs from mapping to mapping - and is an interesting figure of merit. Let us say that on the fermionic lattice we have a hopping term

$$t\ c^{\dagger}_{(i,j)} c_{(i+x,j+y)} + t^{*}\ c^{\dagger}_{(i+x,j+y)} c_{(i,j)}\,, \tag{3.18}$$

where t and t^{*} is a complex coefficient and its Hermitian conjugate. Here the shortest path connecting those modes is over x modes in horizontal and y in vertical direction, the Manhattan distance is $x+y$. Transforming a string with such a distance by one of the three mappings, the connecting string is supported on roughly $O(x+y)$ qubits, but its operator weight is not going to be $x+y$ exactly. In the case of the AQM, we will have twice the number of qubits per mode in the vertical direction, which means that overcoming a vertical distance is more difficult, the string has the weight $x+2y$. In the VCT, the situation is exactly opposite and the horizontal distance is more costly to overcome due to the adjustment costs of the auxiliary modes: the operator weight of the connecting string is $2x+y$. For the BKSF, we find that horizontal and vertical paths are of equal weight, unfortunately the cost is doubled, so $2(x+y)$. Note that different versions of the BKSF exist, where the one version that yields these results is similar to the mapping in [54] - others produce strings of higher weight, for some they are even disconnected.

Note that so far we have omitted the discussion of constant weight overheads, that can arise at the end points of each string, and as such they are just relevant for small Manhattan distances. Around the modes labeled $(i,\, j)$ and $(i+x,\, j+y)$, BKSF and AQM can yield additional terms that matter predominantly for the local hoppings. As discussed, strings in the AQM can have one additional Z-operator around each end-mode, due to costs of the adjustments κ^h_{aux}. In the BKSF, the strings might differ by up to one logical Z-operator on each end, meaning there can be an additional cost of up to three (physical) Z-operators per end. Most notably, the VCT does not have such additional costs making it attractive for the simulation of lattice models, where $x+y$ is small.

Error mitigation - The reduction of the algorithmic depth, that all three mappings aim at, is the main tool in the reduction of noise. However, as the mappings can be regarded as stabilizer codes, it is fair to ask if they can be used for mitigating the effect of noise, as has recently been proposed on a small scale [70, 71]. Intriguingly, the AQM and VCT have local stabilizer tilings that resemble the stabilizers of surface code [17]. However, in contrast to those error correction codes, we cannot achieve topological protection against logical errors. For the planar code of the VCT to correct errors, we necessarily would need the data qubits (the qubits with Z on them in Figure 3.9(d)) to be error free, as X- and Y- errors would masquerade syndromes of errors on the auxiliary qubits. Furthermore, the code cannot detect Z-errors on the data qubits, and even increases their Z-error rate, as syndromes which are stabilizers in surface code differ by some Z-operators from the stabilizers of the VCT. A similar statement can be made for the square lattice AQM, where the auxiliary qubits would have to be perfect, and their X-error rate is increased, see Figure 3.9 (a). Using fewer auxiliary qubits, the square lattice AQM has fewer ears to mitigate errors with (as compared to Figure 3.9(d)), they could however be added with more auxiliary qubits encoding the corresponding horizontal (local) connections. Unlike the surface code, the BKSF (Figure 3.9(c)) has a three-colorable brickwork-pattern in its tiling, that theoretically allows to detect all single-Pauli errors, but like before some weight-two errors tend to masquerade themselves and go undetected when too close together. Although none of the codes allow for topological error correction, they exhibit a limited potential for error mitigation, in which one might be able to catch some errors if the rate is low enough. Whether this is feasible is left to be decided.

In conclusion, although the BKSF has the longest operators, it also requires the fewest qubits. As it is defined on a rotated square lattice, its shape might be the perfect fit for actual devices, as a patch of rotated surface code (including measurement qubits) is a rhombus. The BKSF is probably the most feasible candidate for error mitigation strategies. With its output strings having the lowest weight of all three mappings, the VCT is perhaps the most sophisticated. However, its theoretical backbone is also the most complicated – when using the VCT one would probably have to adhere to the surface-code-like structure of the original proposal. With the weight of the output strings in between the two mappings, AQMs are a compromise for the cases that demand more flexibility. The most unique feature of the AQMs is that we can just use a unitary circuit to promote a data-qubit state into its logical equivalent and if necessary even release it from the auxiliary qubits. The stabilizer state can also be manipulated during the simulation, e.g. accounting for swaps or basis transforms. The state preparation with $V_{\text{aux dat}}$ might make this mapping even interesting for NISQ devices [4], especially for cloud-based quantum computing.

3.8 Conclusion

In this chapter, we have developed a new class of fermion-to-qubit mappings that truly generalize the Jordan-Wigner transform to two dimensions. Moreover, this class can be regarded as a quantum code layer on top of the mapping provided by the Jordan-Wigner transform, and with the unitary $V^{(\dagger)}_{\text{aux dat}}$ we find a means to encode (decode) quantum states in the code layer. The quantum code is shown to require a certain number of auxiliary qubits that is close to N, but this number is not strict. In fact, sparse mappings with a reduced number of auxiliary qubits can achieve similar results, which might be of great practical advantage. More generally, there is a statement that we can make not just about the Auxiliary Qubit Mapping, but also the Verstraete-Cirac transform and the Bravyi-Kitaev Superfast simulation. Versions of all these transforms can be used as one-dimensional linear fermion-to-qubit mapping with N (respectively $N-1$) qubits, but at the expense of additional qubits we can pre-compute certain Pauli strings, which allows us to take shortcuts when mapping operators. This pre-computation is done when said strings are stabilized in a quantum code that entangles data qubits with the qubits added. The usage of these codes allows a quantum computer to do what was not manageable classically: the local treatment of two-dimensional fermion systems. In this way we can not only simulate fermionic lattices, but embed every fermion system on a two-dimensional layout.

We hope that future work will extend these results: we for instance have not taken into account specific limitations on either the qubit connectivity graph or the ability to perform quantum gates, which can be found in proposals for actual devices [18, 72]. It would also be interesting to incorporate the mappings into specific simulation algorithms, to see for instance how phase estimation or qubitization could deal with the planar layout.

3.9 Supplement

3.9.1 Auxiliary Qubit codes

Here we will set up the quantum codes used for the AQMs, which includes the review of the methods developed in [60]. We adapt those methods for quantum codes and contribute ideas which can be used to speed up the initialization of the logical basis.

As mentioned before, the stabilizing the Pauli strings $(p^i_{\text{dat}} \otimes \sigma^i_{N+i})$ effectively describes a quantum code: a larger Hilbert space of $n = N + r$ qubits is constrained to the dimension 2^N by r stabilizer conditions. In contrast to codes for quantum error correction, we do not want to encode information nonlocally, i.e. obtain nonlocal logical operators, but want to localize operators that were nonlocal to begin with. When characterizing a quantum error correction code, one is usually interested in the generating set of stabilizers, the logical basis states, e.g. $|\overline{0}\rangle$, $|\overline{1}\rangle$ and the logical operators, $\overline{X}$, $\overline{Z}$. In the following, we will look at the AQM equivalents of those quantities: while $\{p^i_{\text{dat}} \otimes \sigma^i_{N+i}\}_i$ is a set of stabilizer generators, the extended computational basis $V_{\text{aux dat}} |\boldsymbol{\omega}\rangle_{\text{dat}} \otimes |0^r\rangle_{\text{aux}}$ spans the logical subspace and the adjusted Pauli strings $\widetilde{h}_{\text{aux dat}}$ are its logical operators.

In the initialization of the code space via the unitary $V_{\text{aux dat}}$, the auxiliary qubits are entangled with data qubits, but not before the former are possibly rotated into some basis other than the computational basis: the basis choice of the auxiliary qubits can have consequences for other methods of state preparation and for sure determines the form of the operators σ^i_{N+i} and κ^h_{aux}. In the following, we will introduce the two logical bases, to which AQMs resort. For each of these we will outline the following points:

i. **Logical basis** *Basis of the* $(N+r)$*-qubit states* $|\widetilde{\varphi}\rangle_{\text{aux dat}}$ *with respect to the computational basis* $|\boldsymbol{\omega}\rangle_{\text{dat}}$ *of the* N*-qubit states* $|\varphi\rangle_{\text{dat}}$*.*

ii. **Entangling operation** *The unitary* $V_{\text{aux dat}}$*, for initializing the stabilizer state by quantum gates:* $V_{\text{aux dat}}\,|\varphi\rangle \otimes |0^r\rangle = |\widetilde{\varphi}\rangle_{\text{aux dat}}$*.*

iii. **Hamiltonian adjustments** *Adjustments to be made to Pauli strings,* $h_{\text{dat}} \mapsto h_{\text{dat}} \otimes \kappa^h_{\text{aux}}$*, and adjusted operator mappings.*

We want to deliver the last point in a two-fold way: on the one hand, we present the adjustments to a Hamiltonian in Pauli string form (3.3), where we replace every term $h_{\text{dat}} \mapsto (h_{\text{dat}} \otimes \kappa^h_{\text{aux}})$. The origin of such a Hamiltonian can be arbitrary. On the other hand we want to focus on Hamiltonians that originate from certain many-body problems of fermions. Therefore, we fuse the Hamiltonian adjustments with the linear transform, such that terms $(h_{\text{dat}} \otimes \kappa^h_{\text{aux}})$ can be obtained directly from second quantization as a redefinition of relation (2.12):

$$\begin{aligned} c_j^\dagger &\mathrel{\hat{=}} \frac{1}{2}\left(\bigotimes_{k\in\widetilde{U}(j)} X_k\right)\left(\mathbb{I} + \bigotimes_{l\in\widetilde{F}(j)} Z_l\right)\left(\bigotimes_{m\in\widetilde{P}(j)} Z_m\right)\,, \\ c_j &\mathrel{\hat{=}} \frac{1}{2}\left(\bigotimes_{k\in\widetilde{U}(j)} X_k\right)\left(\mathbb{I} - \bigotimes_{l\in\widetilde{F}(j)} Z_l\right)\left(\bigotimes_{m\in\widetilde{P}(j)} Z_m\right)\,. \end{aligned} \tag{3.19}$$

The redefined transform stays close to the spirit of the original in the sense that only the flip, parity and update sets are replaced by adjusted versions $\widetilde{F}(j)$, $\widetilde{P}(j)$ and $\widetilde{U}(j)$.

Apart from the two bases, we also take a look at an extension of the principle, that allows to build a stabilizer set with strings $\{p^i_{\text{dat}}\}$, that might anticommute. Interestingly, one could in this way encode all terms of a Hamiltonian into a mapping. The resulting code is perhaps most akin to the original method [60], where a new auxiliary qubit is spent for every Hamiltonian term to be multiplied with a stabilizer.

3.9.1.1 Auxiliary qubits in computational basis

With the parity strings being the detrimental substrings of the Jordan-Wigner-transformed Hamiltonians, our main goal is to cancel long strings of Z-operators. In [73], this is achieved in collecting the parity information of subsets of qubits with a circuit QED resonator. In a hardware-unspecific approach, computational basis AQMs store parity information on auxiliary qubits, which can be updated and they have never to be uncomputed.

We generally restrict computational-basis Auxiliary Qubit codes to strings $p^i_{\text{dat}} \subseteq \{\mathbb{I}, Z\}^{\otimes N}$. The p^i_{dat}-strings are here canceled with auxiliary Pauli-Z operators $\sigma^i_{N+i} = Z_{N+i}$. Let us say that the stabilizers are characterized by the $(r \times N)$ binary matrix B, such that an entry '1' in the j-th column on line i of B means that Z_j is part of p^i_{dat}:

$$p^i_{\text{dat}} \otimes \sigma^i_{N+i} \;=\; \left(\bigotimes_{j\in[N]} (Z_j)^{B_{ij}} \right) \otimes Z_{N+i} \,. \tag{3.20}$$

i. Logical basis *In the transformation to a logical state, $|\varphi\rangle_{\text{dat}} \mapsto |\widetilde{\varphi}\rangle_{\text{aux dat}}$, the computational basis is extended to*

$$|\boldsymbol{\omega}\rangle_{\text{dat}} \mapsto |\boldsymbol{\omega}\rangle_{\text{dat}} \otimes |B\boldsymbol{\omega}\rangle_{\text{aux}} \;. \tag{3.21}$$

It is easy to verify that this new basis is stabilized by (3.20) considering $Z_j \,|b\rangle_j \;=\; (-1)^b \,|b\rangle_j$, where $b \in \mathbb{Z}_2$.

ii. Entangling operation *The entangling operation can be described as a (commuting) sequence of* CNOT*-gates that depend on the matrix B. If $B_{ij} = 1$, then there is a* CNOT*-gate in $V_{\text{aux dat}}$, that, controlled on data qubit j, targets the auxiliary qubit labeled $N+i$:*

$$V_{\text{aux dat}} = \prod_{i\in[r]} \;\; \prod_{\substack{j \,\in\, [N] \\ \textit{with } B_{ij} = 1}} \;\; \text{CNOT}\,(j \to N+i) \;. \tag{3.22}$$

The unitary $V_{\text{aux dat}}$, acting on a basis element $(|\boldsymbol{\omega}\rangle_{\text{dat}} \otimes |0^r\rangle_{\text{aux}})$ yields the extended basis of (3.21), considering that
$\text{CNOT}(j \;\to\; k)\,|a\rangle_j \otimes |b\rangle_k \;=\; |a\rangle_j \otimes |a+b\rangle_k$, where $a,\, b \,\in\, \mathbb{Z}_2$. The entangling operation basically stores parity information of subsets of data

qubits (as defined by the rows of B) on auxiliaries. For the exact implementation of $V_{\text{aux dat}}$, (3.22) needs to be adjusted to the connectivity graph of the qubit layout. For square lattice connectivity, the above formula requires $O(rN)$ time steps in the worst case, but there is a way to improve the depth of $V_{\text{aux dat}}$: for the auxiliary qubits i and k, we can replace the circuit

$$\left[\prod_{j:B_{ij}=1} \text{CN}\textsc{ot}(j \to N+i)\right]\left[\prod_{l:B_{kl}=1} \text{CN}\textsc{ot}(l \to N+k)\right] \tag{3.23}$$

$$\begin{aligned}\text{by} \quad & \left[\prod_{j:B_{ij}+B_{kj}=1} \text{CN}\textsc{ot}(j \to N+i)\right] \text{CN}\textsc{ot}(N+k \to N+j) \\ \times \quad & \left[\prod_{l:B_{kl}=1} \text{CN}\textsc{ot}(l \to N+k)\right]. \end{aligned} \tag{3.24}$$

In this (non-commuting) sequence of gates, we let the i-th auxiliary qubit inherit the parity information of the k-th auxiliary qubit by a CNOT-gate inside the aux-register. This is a useful trick when the parity information that is to be stored on these two auxiliary qubits has a large overlap in data qubits, i.e. when the vectors $\bigoplus_x (B_{ix})$ and $\bigoplus_y (B_{ky})$ have a small Hamming distance. In that case, the leftmost product contains only few CNOT-gates, as the bulk of the parity information has been inherited from the $(N+k)$-th qubit.

iii. Hamiltonian adjustments *To maintain the stabilizer state* (3.6)*, we adjust a Pauli string h_{dat} on the data qubits by $h_{\text{dat}} \mapsto (h_{\text{dat}} \otimes \kappa^h_{\text{aux}})$ with*

$$\kappa^h_{\text{aux}} = \bigotimes_{m\in[r]} (X_{N+m})^{\lambda_m}, \tag{3.25}$$

where $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \dots, \lambda_r)^\top \in \mathbb{Z}_2^{\otimes r}$ is obtained by

$$\boldsymbol{\lambda} = \sum_j B\boldsymbol{u_j} \tag{3.26}$$

with $\boldsymbol{u_j}$ being the j-th unit vector of $\mathbb{Z}_2^{\otimes N}$, and the sum extending over all $j \in [N]$, for which h_{dat} acts on the qubit space as X_j or Y_j. Hamiltonian *of adjusted terms* $(h_{\text{dat}} \otimes \kappa^h_{\text{aux}})$ *as in* (3.6) *can be obtained by the redefined transforms* (3.19)*, with the same flip and parity sets,* $\widetilde{F}(j) = F(j)$ *and* $\widetilde{P}(j) = P(j)$*, but the sets* $\widetilde{U}(j)$ *defined from the columns of the matrix*

$$\left[\begin{array}{c} A \\ \hline B \end{array}\right] . \tag{3.27}$$

In case a Pauli string h_{dat} flips a data qubit, that is entangled with a qubit in the aux-register, we have to flip the latter qubit as well. In fact we need to flip all other auxiliaries to which the data qubit contributes: so if we apply the operator X_j to a basis state $|\boldsymbol{\omega}\rangle_{\text{dat}} \otimes |B\boldsymbol{\omega}\rangle_{\text{aux}}$ for $j \in [N]$, we leave the stabilized basis, unless we update the configuration of the auxiliary qubits by $B\boldsymbol{\omega} \mapsto B(\boldsymbol{\omega} + \boldsymbol{u_j})$.

Example

Let us consider a minimal example, in which the data register holds five qubits, and a sixth, an auxiliary qubit, is in the configuration $B\boldsymbol{\omega}$, where B is a (1×5) binary matrix. We consider a Hamiltonian term $h_{\text{dat}} = (X_1 \otimes Z_2 \otimes Z_3 \otimes Z_4 \otimes X_5)$. After adjusting $h_{\text{dat}} \to (h_{\text{dat}} \otimes \kappa^h_{\text{aux}})$, we have the choice to multiply with the stabilizer or not. In Table 3.5 we present the adjusted Hamiltonian before and after multiplication with the stabilizer, considering different choices of B.

B	$h_{\text{dat}} \otimes \kappa^h_{\text{aux}}$	$(h_{\text{dat}} \otimes \kappa^h_{\text{aux}}) \cdot (p^1_{\text{dat}} \otimes Z_6)$
[0 1 0 0 0]	$(X_1 \otimes Z_2 \otimes Z_3 \otimes Z_4 \otimes X_5)$	$(X_1 \otimes Z_3 \otimes Z_4 \otimes X_5 \otimes Z_6)$
[0 1 1 1 0]	$(X_1 \otimes Z_2 \otimes Z_3 \otimes Z_4 \otimes X_5)$	$(X_1 \otimes X_5 \otimes Z_6)$
[1 1 1 0 0]	$(X_1 \otimes Z_2 \otimes Z_3 \otimes Z_4 \otimes X_5 \otimes X_6)$	$-(Y_1 \otimes Z_4 \otimes X_5 \otimes Y_6)$
[1 1 1 1 1]	$(X_1 \otimes Z_2 \otimes Z_3 \otimes Z_4 \otimes X_5)$	$-(Y_1 \otimes Y_5 \otimes Z_6)$

Table 3.5. Adjusted Hamiltonian terms $\widetilde{h}_{\text{aux dat}}$ with respect to the original string $h_{\text{dat}} = (X_1 \otimes Z_2 \otimes Z_3 \otimes Z_4 \otimes X_5)$, depending on the matrix (1×5) matrix B.

3.9.1.2 Auxiliary qubits in Hadamard basis

Extending the idea of [60], we can cancel a set of arbitrary (commuting) strings $\{p^i_{\text{dat}}\}$, where $p^i_{\text{dat}} \in \{X, Y, Z, \mathbb{I}\}^{\otimes N}$, by X-operators: $\sigma^i_{N+i} = X_{N+i}$. Let us characterize the choice of the strings p^i_{dat} by three $(r \times N)$ binary matrices C^X, C^Y and C^Z. Here an entry '1' in C^s_{ji}, with $s \in \{X, Y, Z\}$, indicates that the string p^i_{dat} acts as s on the j-th qubit.

i. Logical basis *In the transformation* $|\varphi\rangle_{\text{dat}} \mapsto |\widetilde{\varphi}\rangle_{\text{aux}\,\text{dat}}$*, the computational basis is extended to*

$$\begin{aligned} |\boldsymbol{\omega}\rangle_{\text{dat}} &\mapsto \left[\prod_{i\in[r]} \frac{1}{\sqrt{2}}\left(\mathbb{I} + p^i_{\text{dat}} \otimes X_{N+i}\right)\right] |\boldsymbol{\omega}\rangle_{\text{dat}} \otimes |0^r\rangle_{\text{aux}} \\ &= \frac{1}{2^{r/2}} \sum_{\boldsymbol{\mu}\in\mathbb{Z}_2^{\otimes r}} \left[\prod_{k\in[r]} \left(p^k_{\text{dat}}\right)^{\mu_k}\right] |\boldsymbol{\omega}\rangle_{\text{dat}} \otimes |\boldsymbol{\mu}\rangle_{\text{aux}} \,. \end{aligned} \tag{3.28}$$

The sum in (3.28) invoke all the possible qubit configurations $\boldsymbol{\mu} \in \mathbb{Z}_2^{\otimes r}$ with equal weight. This is a result of the auxiliary qubits being in Hadamard basis. This choice of basis becomes plausible by multiplying a basis state (3.28) with one of the stabilizers $(p^i_{\text{dat}} \otimes X_{N+i})$:

$$\begin{aligned} &\left(p^i_{\text{dat}} \otimes X_{N+i}\right) \frac{1}{2^{r/2}} \sum_{\boldsymbol{\mu}\in\mathbb{Z}_2^{\otimes r}} \left[\prod_{k\in[r]} \left(p^k_{\text{dat}}\right)^{\mu_k}\right] |\boldsymbol{\omega}\rangle_{\text{dat}} \otimes |\boldsymbol{\mu}\rangle_{\text{aux}} \\ &\qquad = \frac{1}{2^{r/2}} \sum_{\boldsymbol{\mu}\in\mathbb{Z}_2^{\otimes r}} \left[\prod_{k\in[r]} \left(p^k_{\text{dat}}\right)^{\mu_k+\delta_{ik}}\right] |\boldsymbol{\omega}\rangle_{\text{dat}} \otimes |\boldsymbol{\mu} + \boldsymbol{u_i}\rangle_{\text{aux}} \,. \end{aligned} \tag{3.29}$$

If we now shift the binary vector in the sum by the i-th unit vector $\boldsymbol{u_i}$ to $\boldsymbol{\mu} \mapsto \boldsymbol{\mu} + \boldsymbol{u_i}$, the original basis element on the right-hand side of (3.28) is recovered and thus the set of Pauli strings $(p^i_{\text{dat}} \otimes X_{N+i})$ stabilizes every state $|\widetilde{\varphi}\rangle_{\text{aux}\,\text{dat}}$ that is in the subspace spanned by (3.28).

ii. Entangling operation *Following [60], the entangling operation can be described as*

$$V_{\text{aux}\,\text{dat}} = \prod_{i\in[r]} \left(|0\rangle\langle 0|_{N+i} + p^i_{\text{dat}} \otimes |1\rangle\langle 1|_{N+i}\right) \mathrm{H}_{N+i} \,, \tag{3.30}$$

where H_{N+i} *is the Hadamard gate on the* $(N+i)$*-th qubit. In words,* $V_{\text{aux}\,\text{dat}}$ *can be realized by a unitary quantum circuit that first applies Hadamard gates to every auxiliary qubit, and then applies each string* p^k_{dat} *controlled on the* k*-th auxiliary qubit.*

We notice that the circuit (3.30), when acting on a state $|\varphi\rangle_{\text{dat}} \otimes |0^r\rangle_{\text{aux}}$, firstly changes the basis of the auxiliary register into

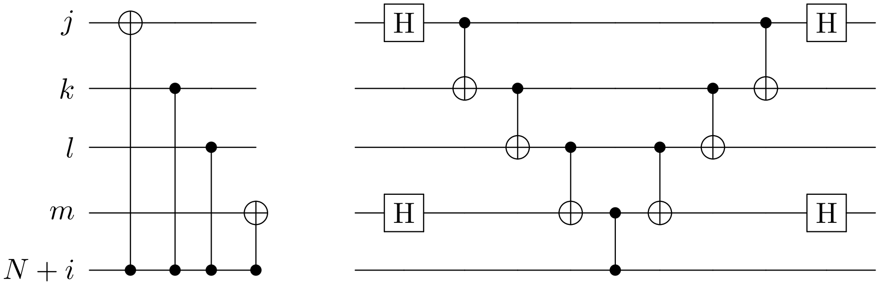


Figure 3.10. Two versions of the controlled application of the Pauli string $p^i_{\text{dat}} = (X_j \otimes Z_k \otimes Z_l \otimes X_m)$ on the i-th qubit in the auxiliary register.

$|+^r\rangle_{\text{aux}} = (\bigotimes_{i\in[r]} |+\rangle_{N+i}) = r^{-\frac{1}{2}} \sum_{\boldsymbol{\mu}\in\mathbb{Z}_2^{\otimes r}} |\boldsymbol{\mu}\rangle_{\text{aux}}$. Then the controlled application of the strings p^i_{dat} entangles auxiliary and data qubits. In principle, this can be done by CNOT, CPHASE and controlled-Y gates according to the action of a string p^i_{dat} on each data qubit, see Figure 3.10 (left). In practice, the required qubit connectivity might however not be available, such that we may resort to an implementation of the circuit as in Figure 3.10 (right). Like for the codes with computational-basis auxiliary qubits, we can here apply tricks to make $V_{\text{aux dat}}$ more shallow whenever two strings $p^i_{\text{dat}}, p^k_{\text{dat}}$ are similar to one another: after the Hadamard-gates are applied to the auxiliary qubits i and k, we can replace the circuit

$$\Big(|0\rangle\langle 0|_{N+i} + p^i_{\text{dat}} \otimes |1\rangle\langle 1|_{N+i}\Big)\ \Big(|0\rangle\langle 0|_{N+k} + p^k_{\text{dat}} \otimes |1\rangle\langle 1|_{N+k}\Big) \quad \text{by} \tag{3.31}$$

$$\Big(|0\rangle\langle 0|_{N+i} + \Big(p^i_{\text{dat}} \cdot p^k_{\text{dat}}\Big) \otimes X_{N+k} \otimes |1\rangle\langle 1|_{N+i}\Big)\ \Big(|0\rangle\langle 0|_{N+k} + p^k_{\text{dat}} \otimes |1\rangle\langle 1|_{N+k}\Big)\ , \tag{3.32}$$

which means that instead of applying the string p^i_{dat}, we conditionally apply the string that results from the operator product of p^i_{dat} with p^k_{dat}, and an X-operator on the k-th auxiliary qubit. What we use here is the fact that the $(N+k)$-th qubit is already entangled with the data qubits after the right sequence of controlled gates, such that we can use the stabilizer condition in the sequence on the left. For this to work, the order in which the two resulting strings are initialized is now fixed. A minus sign that might occur in the operator product can be reproduced by adding a Z_{N+i}, [42]. Before presenting the Hamiltonian adjustments, it is left for us to verify that the controlled applications of p^i_{dat} on $|\boldsymbol{\omega}\rangle_{\text{dat}} \otimes |+^r\rangle_{\text{aux}}$ yield the corresponding element of the extended basis (3.28). Let us consider the following reformulation of the controlled-(p^i_{dat}) terms:

$$\begin{aligned}\prod_{i\in[r]} &\left(|0\rangle\langle 0|_{N+i} + p^i_{\text{dat}} \otimes |1\rangle\langle 1|_{N+i}\right) \\ &= \prod_{i\in[r]} \left(\sum_{\mu'_i \in \mathbb{Z}_2} (p^i_{\text{dat}})^{\mu'_i} \otimes |\mu'_i\rangle\langle \mu'_i|_{N+i} \right) \\ &= \sum_{\boldsymbol{\mu'} \in \mathbb{Z}_2^{\otimes r}} \left[\prod_{k\in[r]} \left(p^k_{\text{dat}}\right)^{\mu'_k} \right] \otimes |\boldsymbol{\mu'}\rangle\langle \boldsymbol{\mu'}|_{\text{aux}} \,. \end{aligned} \tag{3.33}$$

Considering the expansion of $|+^r\rangle_{\text{aux}}$ in the computational basis, we can proceed to arrive at (3.28) by inspection.

iii. Hamiltonian adjustments *For a Pauli string h_{dat} to maintain the stabilizer state* (3.6)*, we adjust it by*

$$\kappa^h_{\text{aux}} = \bigotimes_{j\in T(h)} Z_{N+j} \,, \tag{3.34}$$

where the set $T(h) \subseteq [r]$ contains k if p^k_{dat} anticommutes with h_{dat}. As a consequence, a Hamiltonian of terms $(h_{\text{dat}} \otimes \kappa^h_{\text{aux}})$ can be obtained from second quantization using the redefined transformations (3.19)*, where the update sets are defined as before $\widetilde{U}(j) = U(j)$, but flip and parity sets $\widetilde{F}(j)$, $\widetilde{P}(j)$ are redefined by the rows of the matrices*

$$\left[A \,\middle|\, C^X + C^Y\right] \,, \quad \left[RA \,\middle|\, R(C^X + C^Y) + C^Y + C^Z\right] \,. \tag{3.35}$$

We will now show that the adjusted Pauli string $(h_{\text{dat}} \otimes \kappa^h_A)$ acts on a state $|\widetilde{\varphi}\rangle_{\text{aux dat}}$ such that after application of $V^\dagger_{\text{aux dat}}$, we recover $h_{\text{dat}}\,|\varphi\rangle_{\text{dat}} \otimes |0^r\rangle_{\text{aux}}$. We start by applying the adjusted term to the extended state. The goal is to use (anti-)commutation relations with the strings p^k_{dat} to let h_{dat} act on the data register first. It turns out that minus signs that we pick up by anticommutations are exactly canceled by sign changes originating from κ^h_A acting on the aux-register.

In general, we find if h_{dat} now anticommutes with a string p^k_{dat}, then $k \in T(h)$ such that $(h_{\text{dat}} \otimes \kappa^h_{\text{aux}})$ commutes with $(\mathbb{I} + p^k_{\text{dat}} \otimes X_{N+k})$, and we find (3.5) satisfied. For the transform (3.35), we take into account all sorts of Pauli operators that originate from parity, update and flip operators, by which we mean the strings $(\bigotimes_{m\in P(j)} Z_m)$, $(\bigotimes_{k\in U(j)} X_k)$ and

$(\bigotimes_{l\in F(j)} Z_l)$ in (2.12). If X- and Y-operators in a string p^i_{dat} anticommute with the Z-operators in the j-th flip operator, we have to counteract by adjusting it with a Z-operator on the i-th auxiliary: $(\bigotimes_{l\in F(j)} Z_l) \otimes Z_{N+i}$. The same argument holds for the parity operators, but we also add Z-operators there, stemming from anticommutations of the update operator with Z- and Y-operators in p^i_{dat}. Considering that the operators X, Y and Z appear in the strings p^i_{dat} according to the C-matrices, we can use these matrices to describe the contents of the flip and parity sets, by which we obtain (3.35).

Example

As an example we examine a 5-qubit Hamiltonian term, $h_{\text{dat}} = (X_1 \otimes Z_2 \otimes Z_3 \otimes Z_4 \otimes X_5)$. The sixth qubit is a Hadamard-basis auxiliary, used to cancel various substrings p^1_{dat}. In Table 3.6, we find the adjusted terms $(h_{\text{dat}} \otimes \kappa^h_{\text{aux}})$ and the deformed terms, $(p^1_{\text{dat}} \otimes X_6) \cdot (h_{\text{dat}} \otimes \kappa^h_{\text{aux}})$, for various choices of the stabilizer $(p^1_{\text{dat}} \otimes X_6)$.

p^1_{dat}	$(h_{\text{dat}} \otimes \kappa^h_{\text{aux}})$	$(p^1_{\text{dat}} \otimes X_6) \cdot (h_{\text{dat}} \otimes \kappa^h_{\text{aux}})$
$(Z_2 \otimes Z_3 \otimes Z_4)$	$(X_1 \otimes Z_2 \otimes Z_3 \otimes Z_4 \otimes X_5)$	$(X_1 \otimes X_5 \otimes X_6)$
$(X_1 \otimes Z_2 \otimes Z_3 \otimes X_4)$	$(X_1 \otimes Z_2 \otimes Z_3 \otimes Z_4 \otimes X_5 \otimes Z_6)$	$-(Y_4 \otimes X_5 \otimes Y_6)$
$(X_1 \otimes Z_2 \otimes Z_3 \otimes Z_4 \otimes X_5)$	$(X_1 \otimes Z_2 \otimes Z_3 \otimes Z_4 \otimes X_5)$	X_6

Table 3.6. Adjusted Hamiltonians $\widetilde{h}_{\text{aux dat}}$ to $h_{\text{dat}} = (X_1 \otimes Z_2 \otimes Z_3 \otimes Z_4 \otimes X_5)$, depending on the choice of p^1_{dat}.

3.9.1.3 Stabilizing anticommuting data-qubit strings

We present a more general quantum code based on auxiliary qubits in Hadamard basis, but in which the strings $\{p^i_{\text{dat}}\}$ do not necessarily have to commute. Using this code, an entire Hamiltonian can in principle be transformed into interactions on only the auxiliary qubits. The general idea here is to amend the scheme by the following notion: in order to counter anticommutations, we replace the (single-qubit) Pauli operators σ^i_{N+i} with Pauli strings on the auxiliary register γ^i_{aux}, such that γ^i_{aux} contains X_{N+i} as before, but for every other string p^k_{dat} with $k < i$, that anticommutes with p^i_{dat}, it contains a Z-operator, Z_{N+k}. For convenience we define the operation $\star$ as:

$$i \star k = \begin{cases} 0 & \text{if} \quad [p^i_{\text{dat}}, p^k_{\text{dat}}] \;\; = 0 \\ 1 & \text{if} \quad [p^i_{\text{dat}}, p^k_{\text{dat}}]_+ = 0 \end{cases} . \tag{3.36}$$

Using this notation, we define the stabilizers of our system as

$$p^i_{\text{dat}} \otimes \gamma^i_{\text{aux}} \;=\; p^i_{\text{dat}} \otimes \left(\bigotimes_{k \in [i-1]} (Z_{N+k})^{i \star k} \right) \otimes X_{N+i} \,, \tag{3.37}$$

since all Pauli strings $(p^i_{\text{dat}} \otimes \gamma^i_{\text{aux}})$ have to commute pairwise for all $i \in [r]$ as defined above. We will now turn to describe the mapping in the established way.

i. Extended basis *The computational basis $|\boldsymbol{\omega}\rangle_{\text{dat}}$ is extended to:*

$$|\boldsymbol{\omega}\rangle_{\text{dat}} \;\mapsto\; \frac{1}{2^{r/2}} \sum_{\boldsymbol{\mu} \in \mathbb{Z}_2^{\otimes r}} \left[(p^1_{\text{dat}})^{\mu_1} \;\cdots\; (p^r_{\text{dat}})^{\mu_r} \right] |\boldsymbol{\omega}\rangle_{\text{dat}} \otimes |\boldsymbol{\mu}\rangle_{\text{aux}} \,. \tag{3.38}$$

This basis resembles (3.28), with the subtle difference that the order of the strings p^i_{dat} matters here. When stabilizer $(p^i_{\text{dat}} \otimes \gamma^i_{\text{aux}})$ are multiplied to (3.38) from the right, the operators γ^i_{aux} cancel all minus signs from anticommutations, and flip the i-th qubit in the auxiliary register. Note that the order of the strings p^i_{dat} in (3.38) is to be taken into account when we attempt to encode $|\widetilde{\varphi}\rangle_{\text{aux dat}}$ from $|\varphi\rangle \otimes |0^r\rangle_{\text{aux}}$.

ii. Entangling operation *We pick a sequence $i_1, i_2, ..., i_r$ that is some permutation of $1, 2, ..., r$, in which we want to perform the entangling operation for the stabilizers $(p^{i_m}_{\text{dat}} \otimes \gamma^{i_m}_{\text{aux}})$, where the stabilizer of number i_r is taken care of first, and the one labeled i_1 last. The entangling operation associated with that sequence is*

$$V_{\text{aux dat}} \;=\; \prod_{m=1}^{r} \left(|0\rangle\langle 0|_{N+i_m} + p^{i_m}_{\text{dat}} \otimes |1\rangle\langle 1|_{N+i_m} \otimes \left[\bigotimes_{k>m} (Z_{N+i_k})^{(i_m \,\star\, i_k)\; \theta_{i_m i_k}} \right] \right) \mathrm{H}_{N+i_m} \,, \tag{3.39}$$

Note that if the we pick the original order, $i_m = m$, the circuit almost looks like (3.30), but, again, here the exact order matters. The Hamiltonian adjustments are identical to (3.34) and (3.35), as the only difference, the ordering of the strings p^i_{dat}, does not matter there: a Hamiltonian term

$\widetilde{h}_{\text{aux dat}}$ needs to pass all p^k_{dat} in (3.38), picking up all minus signs possible. We have thus obtained an auxiliary qubit mapping with completely arbitrary set of strings p^i_{dat}. If this string is a Hamiltonian term $h_{\text{dat}} = p^i_{\text{dat}}$, we can eliminate its action on the data qubits by replacing

$$h_{\text{dat}} \;\mapsto\; \left(h_{\text{dat}} \otimes \kappa^h_{\text{aux}}\right) \cdot \left(p^i_{\text{dat}} \otimes \gamma^i_{\text{aux}}\right) \;=\; X_{N+i} \otimes \left[\bigotimes_{k>i} \left(Z_{N+k}\right)^{i\star k}\right] . \tag{3.40}$$

The entire Hamiltonian can in this way be pre-computed and reduced to an action on only the auxiliary register.

3.9.2 Tree-based transforms

In this section, we consider fermion-to-qubit mappings defined on tree structures for a setup with limited connectivity. This particular class of mappings is part of the mappings considered in Section 2.3 (so $n = N$), where the tree structures are inherent in the definition of the transformation matrix A. Although this class technically contains the Jordan-Wigner transform, our motivation is to obtain mappings that are more akin to the Bravyi-Kitaev transform, in order to keep parity strings short. While the Bravyi-Kitaev transform itself does this job perfectly, we will show that it cannot be reconciled with a square lattice connectivity graph: in this section, we instead develop a method to tailor mappings to preexisting connectivity graphs, and provide an algorithm with which short parity strings can be guaranteed and the operator weight bounded. Let us start by reviewing the Bravyi-Kitaev transform.

In [26], the mapping is introduced in order to reduce the weight of transformed fermionic operators to $O(\log N)$, which is an exponential improvement over the Jordan-Wigner transform. In the original paper, the (classical) encoding and decoding are defined by a partially ordering the mode indices according to some rules defined by their representation as binary numbers. Later works then developed the notion of flip, update and parity sets and provided a method to construct the binary matrices A^{-1} and A in $\log N$ steps [27, 37]. Instead of being one-dimensional, the partial order can be regarded placing all mode indices onto nodes inside a tree structure, which is the reason the mapping is sometimes referred to as binary-tree transform (even though the tree is not a binary tree). As pointed out in [29], the flip and update operators of every mode j, $(\bigotimes_{k\in F(j)} Z_k)$ and $(\bigotimes_{l\in U(j)} X_l)$, have a geometric interpretation on that tree (as will be illustrated shortly), so we would naturally like to match it with the qubit-connectivity graph. While an embedding is possible for small such trees, increasing N will make the tree outgrow the square lattice rather quickly. In fact, the binary rule implies that the node with index 2^j has exactly j children, and all nodes with indices below 2^j have fewer than j children. This means that trees with $N > 16$ modes, cannot be embedded in the square lattice where every site has 4 nearest neighbors. The tree for $N = 16$ can be found in Figure 3.11(a) and its embedding in the square lattice is presented in panel (b). This particular tree is however not the end of the story. In [29], it was argued that the Bravyi-Kitaev transform can be optimized to produce more local strings, in particular when considering Hamiltonians of locally-interacting fermions. For that purpose, the 'binary' trees are replaced with segmented Fenwick-tree structures. These structures are explicitly allowed to contain multiple trees, and the number of trees is even a parameter of the mapping. This number can range from 1 to N (the number of modes), where at N the mapping is identical to the Jordan-Wigner transform and at 1 it corresponds the Bravyi-Kitaev transform (in case N is an integer power of two). However, we can go even further and define mappings based on an arbitrary number of arbitrary trees. In particular, we can define tree structures that can be embedded on arbitrary qubit connectivity graphs, like our square lattice, and the associated mappings still yield small parity operators ($\bigotimes_{m \in P(j)} Z_m$). Let us consider one specific connectivity graph.

We need to pick a forest (a set of trees) which in total has a number of N nodes. As each node will correspond to one qubit, the trees need to be connected to each other, and so we connect their respective roots. It is sufficient here for each root to be connected to two others, such that they are linked like a chain with their order foreshadowing some canonical ordering. We now choose a set of trees, such that the graph created by connecting them can be embedded in the actual qubit-connectivity graph. Let us now turn to the description of the mapping itself. For that purpose, we firstly need to assign an index to every node, a process for which we later will provide an algorithm, but for now let us assume we have done so in a prudent way. For the definition of the transform, it is sufficient to give a definition of all update and flip sets, as by corresponding sets $F(j)$ and $U(j)$ the matrices A and A^{-1} can be inferred column- and row-wise. For the flip set of index j, $F(j)$, we consider the node with index j and all its children in the tree it is on, i.e. all the nodes directly connected to j on edges that lead away from the root. The update set $U(j)$ includes the node j and all its ancestors, i.e. all nodes on the direct line to the root (of the tree it is on), where the root is also included. A visual representation of these operators can be found in Figure 3.11(c), where the direction with respect to the root is indicated by arrows. Their embedded version can be found in panel (d) of the figure. Note that this means that by the encoding of this mappings, qubit j stores the parity information of mode j and all other modes whose index is beneath j in the tree.

For anticommutation relations like $[c_i, c_j^\dagger]_+ = \delta_{ij}$, it is important that

$$\left(\bigotimes_{k\in F(i)} Z_k\right)\left(\bigotimes_{l\in U(j)} X_l\right) = (-1)^{\delta_{ij}}\left(\bigotimes_{l\in U(j)} X_l\right)\left(\bigotimes_{k\in F(i)} Z_k\right), \quad (3.41)$$

which we now want to verify by the definitions of the flip and update sets. If j is any descendant of i, then the two operators overlap on two qubits, which means they commute. If it is not an ancestor, then the only case where the operators have overlap is when $i = j$, where they exactly overlap on that very qubit and anticommute.

We so far have suppressed the discussion of the parity operators, that will now lead into an algorithm for the index assigning and a bound for the operator weight. Let us assume that our forest consisted of τ trees, each of which has at most Λ levels and every node at most Γ children. We know that the operator weight of update and flip operators scales as $O(\Lambda + 1)$ and $O(\Gamma + 1)$, the structure of the parity set however now depends on the index assigned to the nodes. By a binary rule, the Bravyi-Kitaev transform manages to only involve $O(\log N)$ qubits in the parity operators, and we can devise a labeling that mirrors its principle. The parity operator of j is only the product of flip operators of $i < j$. On the other hand, multiplying the flip operator of a parent node k with all flip operators of its descendants will cancel all Z-operators but Z_k. Thus, in order for the parity operator of j to have low weight, as many nodes with labels $i < j$ as possible need to be descendants of j. Subsequently, the mapping with the smallest parity sets is characterized by a tree where every node has only one child, i.e. a vertical line. This mapping, that we recall as parity transform from [27], has however the problem of $O(N)$-weight update operators, and is thus of the same quality as the Jordan-Wigner transform. Indeed, one being characterized by a vertical line, the

other by a horizontal line (connected one-node trees), makes both mappings effectively one-dimensional. In order to minimize the weight of update and parity operators altogether, we need to reconcile the cancellation strategy with the tree structure. The idea is to involve only qubits in $P(j)$, that are children of the nodes in $U(j)$. Of course, this is not quite possible. If an entire tree only contains nodes $i < j$, then $P(j)$ will always contain the root of this tree. According to the formula (2.12), transforming $c_j^{(\dagger)}$ thus results in strings of weight $O(\tau + \Lambda\Gamma)$. Not only this, but the strings produced will also be continuous for transforms of single operators. Unfortunately, for pairs of operators like $c_i^\dagger c_j$, the strings are discontinuous on the first qubit that is both, an ancestor of i and j – a situation we cannot remedy.
The question is now how to assign the labels to the nodes such that this mapping is implemented, or in other words: given an unlabeled forest with connected roots, how can we obtain a mapping that outputs strings of weight $O(\tau + \Lambda\Gamma)$? For that purpose, we put labels 1 to N (in order) on the nodes according to the little program below.

Line 1 Consider the first tree in line.

Line 2 Choose a leaf and put a label on it.

Line 3 Check whether there are unlabeled siblings. If it does, choose such a sibling for the consideration in the following step. If not, proceed to Line 5.

Line 4 Check whether the current node is a leaf, and if it is, label it, otherwise put a label on a leaf chosen from the sub-tree of which the current node is the root. Continue from Line 3 with the last-labeled node.

Line 5 Check whether the last node considered has a parent. If there is a parent, put a label on it and continue from Line 3 with it. In case there is none, the previous node was a root, and we label it and proceed with the next line.

Line 6 If the root is the top of the last tree, the program ends, but if it is not, the next tree in line is considered and the program continues from Line 2.

By the end of the program, all nodes are labeled in a way such that the resulting mapping outputs strings of weight $O(\tau + \Lambda\Gamma)$. Note that there might be variations on how this process can turn out, since in several lines an element of choice is involved. We can now consider customized trees and root-connected forests. For instance, we can consider a perfect binary tree (a real one this time), which yields a $O(\log N)$ scaling as well. Although with such a tree, every node is only required to have three nearest-neighbors, the embedding of an arbitrarily-sized tree into a square lattice is still not possible. This is due to the children that run into each other as we expand the tree-embedding on the lattice. We hope however that for future work the tools provided in this section will help to tailor tree-based transforms directly to specific device layouts.

3.9.3 Technical details

This section is dedicated to the quantum codes in the foundation of every locality-preserving fermion-to-qubit mapping referenced in this chapter. In particular, we provide details on the features that distinguish our mappings from prior works: rather than trying to mimic the locality of the simulated system, we have focused on the quantum device and encoded fermions into local terms on its connectivity. However, since the fermionic Hamiltonian is local on a different graph, we have introduced Manhattan-distance strings to tackle this mismatch. To comply with those ideas, we have chosen the quantum codes to be planar on the square lattice, with locality and Manhattan-distance properties reflected in their logical operators. Catching up on a number of technical details omitted earlier in the text, we commence this section by showing the latter for the logical operators of the square lattice AQM. Specifically, we have claimed that each (relevant) logical operator only had small support on the auxiliary qubits – a statement we we will substantiate in Section 3.9.3.1 by decomposing fermion operators into Majoranas. Note that those Majorana operators should not be regarded as physical particles, but rather as a useful description of the model at hand. With this new tool, we then motivate the Manhattan-distance property in Section 3.9.3.2. Afterwards, we turn to BKSF and VCT in Sections 3.9.3.4 and 3.9.3.3. We review the literature implementations of those mappings and then adapt them such that the codes are planar on the square lattice layout. What is more, we show that our implementation results in logical operators with the Manhattan-distance property. We also add some points about the proper code space

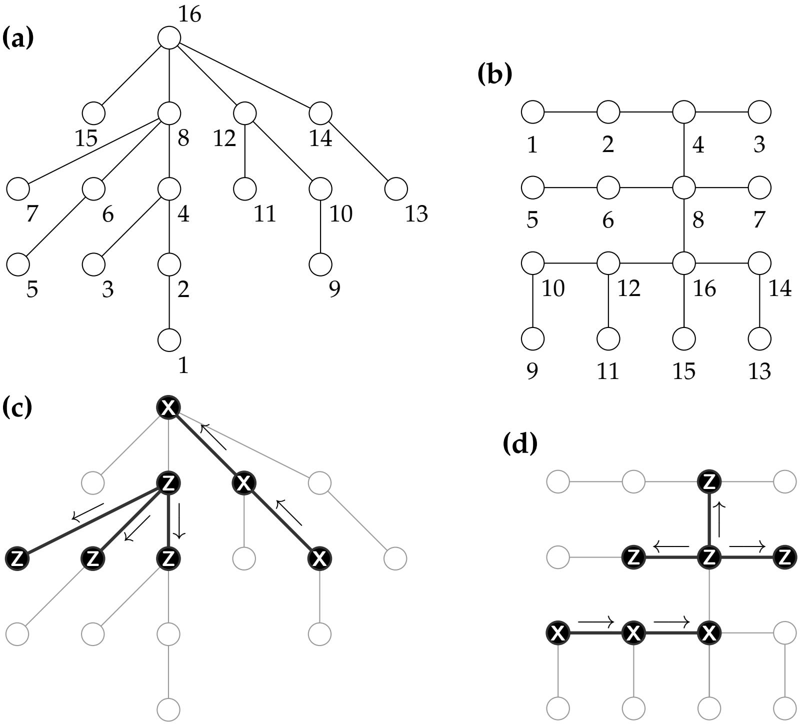


Figure 3.11. **(a)** Tree of the Bravyi-Kitaev transform for 16 qubits. Qubits are labeled from 1 to 16 according to the underlying binary tree rule. **(b)** Embedding the tree of 16 qubits into a (4×4) square lattice. **(c) & (d)** Pauli strings $(\bigotimes_{i\in U(10)} X_i)$ and $(\bigotimes_{i\in F(8)} Z_i)$ on the tree and the square lattice, where the arrows indicate the rules that determine the update set $U(10)$, and the flip set $F(8)$ respectively: $F(i)$ would involve node i and all its children, whereas $U(j)$ would involve involves node j and all its ancestors including the root.

and the logical basis, which is relevant for the logical state preparation and transformation of the Hamiltonian. For the BKSF, we describe how to constrain the simulated parity sector, and discuss the subspace of the auxiliary system for the VCT. Lastly, we consider both mappings applied to the Hubbard model, showing some of the strings referenced in Table 3.4.

3.9.3.1 Auxiliary Qubit support of the square lattice AQM

Majorana particles are fermions as their many-body wave-functions are anti-symmetric under permutation. Majorana operators $m_j^{(\dagger)}$ thus satisfy anticommutation relations like (1.7), but they are also their own antiparticles, making the operators Hermitian: $m_j^\dagger = m_j$. In general, these operators describe the relations

$$[m_i,\, m_j]_+ \;=\; 2\delta_{ij} \quad \text{and} \quad m_i m_i = 1\,. \tag{3.42}$$

For each fermionic mode, we need two Majoranas, such that the fermionic operators $c_j^{(\dagger)}$ are described by two Majorana species m_j and $\overline{m}_j$, where $\overline{m}_j$ obey the same relations (3.42), and are indistinguishable to m_j, which means $m_i\,\overline{m}_j \;=\; -\overline{m}_j\, m_i$. We define

$$c_j^\dagger = \frac{1}{2}\left(m_j - i\,\overline{m}_j\right) \quad \text{and} \quad c_j = \frac{1}{2}\left(m_j + i\,\overline{m}_j\right)\,. \tag{3.43}$$

Thus we can represent the operators m_j, $\overline{m}_j$ with the Jordan-Wigner transform as

$$m_j \,\hat{=}\, \left(\bigotimes_{k=1}^{j-1} Z_k\right) \otimes X_j \quad \text{and} \quad \overline{m}_j \,\hat{=}\, \left(\bigotimes_{k=1}^{j-1} Z_k\right) \otimes Y_j\,. \tag{3.44}$$

Keeping the canonical order in mind, we turn mode and qubit indices once again into coordinates just like in Section 3.5. With the index j being found at coordinate $\boldsymbol{R} = (R_1,\, R_2)^\top$, the two Pauli strings (3.44) will be denoted by $\mathcal{M}_{\text{dat}}^{b,\boldsymbol{R}}$, where $b \in \mathbb{Z}_2$ with $\mathcal{M}_{\text{dat}}^{0,\boldsymbol{R}} \,\hat{=}\, m_{\boldsymbol{R}}$ and $\mathcal{M}_{\text{dat}}^{1,\boldsymbol{R}} \,\hat{=}\, \overline{m}_{\boldsymbol{R}}$. It is important to note that H_{dat} is comprised of strings $\mathcal{M}_{\text{dat}}^{b,\boldsymbol{R}}$ in the same way the fermionic Hamiltonian is constructed from products of $m_{\boldsymbol{R}}$ and $\overline{m}_{\boldsymbol{R}}$. As a Hamiltonian of the form (1.8) only features products of at most four Majorana operators, the same can be said about Jordan-Wigner transformed Hamiltonians and the $\mathcal{M}$-strings. Therefore we only have to put a bound on the weight gained in the adjustment process $h_{\text{dat}} \mapsto h_{\text{dat}} \otimes \kappa_{\text{aux}}^h$ for any $h_{\text{dat}} = \mathcal{M}_{\text{dat}}^{b,\boldsymbol{R}}$. Since the adjustment is a linear process, the total weight that any term gains is four-fold that of a single $\mathcal{M}$-string. Indeed, for a coordinate $\boldsymbol{R}$ in the bulk of the qubit array, we find

$$\mathcal{M}_{\text{dat}}^{b,\boldsymbol{R}} \;\mapsto\; \mathcal{M}_{\text{dat}}^{b,\boldsymbol{R}} \otimes Z_{\boldsymbol{R}\pm\frac{1}{2}\boldsymbol{e_2}}\,, \tag{3.45}$$

where $\boldsymbol{e_2} = (0,\, 1)^\top$ is the Cartesian unit vector in vertical direction, and we recall that auxiliary are placed at half integer positions of the vertical coordinate. The point is that the adjustments only include information of one of the two auxiliaries adjacent to the data qubit at $\boldsymbol{R}$. As claimed before, this means that for hopping terms, i.e. $c_i^\dagger c_j$, the adjustments are local at the end points of the resulting strings. Note that in the case where $\boldsymbol{R}$ is at the boundary of the qubit array, there is only an adjustment such as in (3.45) if the corresponding auxiliary qubit exists.

Let us now briefly illustrate the statement (3.45). We first express the stabilizers (3.9) in terms of $\mathcal{M}_{\text{dat}}^{b,\boldsymbol{R}}$. For a stabilizer $(p_{\text{dat}}^{\boldsymbol{R}+\frac{1}{2}\boldsymbol{e_2}} \otimes X_{\boldsymbol{R}+\frac{1}{2}\boldsymbol{e_2}})$, we find

$$p_{\text{dat}}^{\boldsymbol{R}+\frac{1}{2}\boldsymbol{e_2}} = \begin{cases} i\mathcal{M}_{\text{dat}}^{0,\boldsymbol{R}} \cdot \mathcal{M}_{\text{dat}}^{0,\boldsymbol{R}+\frac{1}{2}\boldsymbol{e_2}} & \text{if } R_2 \text{ is odd} \\ -i\mathcal{M}_{\text{dat}}^{1,\boldsymbol{R}} \cdot \mathcal{M}_{\text{dat}}^{1,\boldsymbol{R}+\frac{1}{2}\boldsymbol{e_2}} & \text{if } R_2 \text{ is even}\,. \end{cases} \tag{3.46}$$

From $\mathcal{M}_{\text{dat}}^{b,\boldsymbol{R}}\,\mathcal{M}_{\text{dat}}^{a,\boldsymbol{S}} = (-1)^{1+\delta_{\boldsymbol{SR}}\,\delta_{ab}}\mathcal{M}_{\text{dat}}^{a,\boldsymbol{S}}\,\mathcal{M}_{\text{dat}}^{b,\boldsymbol{R}}$ it is apparent that a physical operator $\mathcal{M}_{\text{dat}}^{b,\boldsymbol{R}}$ can only anticommute with a p-strings local to $\boldsymbol{R}$ or $\boldsymbol{R} - \frac{1}{2}\boldsymbol{e_2}$ (so they both exist). Since the species index has to match as well, only one of the two stabilizers will anticommute and make an adjustment (3.45) necessary. Note that it hinges on both b and the vertical component of $\boldsymbol{R}$ whether that adjustment is $Z_{\boldsymbol{R}+\frac{1}{2}\boldsymbol{e_2}}$ or $Z_{\boldsymbol{R}-\frac{1}{2}\boldsymbol{e_2}}$ in particular.

3.9.3.2 Manhattan-distance property

Verstraete-Cirac transform, Superfast simulation and the square lattice AQM - all three mappings inherently posses the Manhattan-distance property, which means that when we use them to transform hopping interaction of two fermionic modes, the weight of the (shortest) resulting Pauli string can be bounded with the Manhattan distance of the modes on the fermionic lattice. Here we will show that all mappings work in a similar fashion that enables us to use this property and elucidate why it is necessary to make use of it in a limited qubit layout. Let us recall the definition of the Majorana fermions from Section 3.9.3.1.

Majorana-pair operators like im_jm_k are used in the original proposals of VCT and BKSF, and their structure is also an element in the AQM. This is because these operators can be transformed into single Pauli strings that describe the interaction of two fermionic modes j and k, making them a useful tool for modeling it. As already established, they also have quite convenient (anti-) commutation relations. All mappings introduce extra qubits to encode operators corresponding to Majorana pairs $(m_j\, m_k) \hat{\propto} \mathcal{O}_{jk}$. In one or the other way, all mappings use these operators to prevent hopping terms, as they occur in fermionic Hamiltonians, to become nonlocal Pauli strings in the qubit Hamiltonian. When nonlocal connections of modes i with k, and k with j, as well as i with j appear in a fermionic Hamiltonian, one might think of encoding three operators $\mathcal{O}_{ik}$, $\mathcal{O}_{kj}$ and $\mathcal{O}_{ij}$. However, all mappings exhibit repercussions for adding qubits to encode these operators, such as a weight increase in the substrings κ^h_{aux} in case of the AQM. There is also the issue that we need to connect all the modes in a way that would mimic the connectivity graph of the fermionic Hamiltonian - a Hamiltonian that is generally more complicated than a lattice model. In order to be modest with the amount of qubits to be added and to be able to deal with the limited connectivity of the setup, we reconsider encoding operators $\mathcal{O}_{ij}$ of all possible combinations ij by adding qubits. Instead, under the cost of a slightly higher operator weight, we can obtain some nonlocal $\mathcal{O}_{ij}$ by multiplying operators that are already encoded: $\mathcal{O}_{ij} \propto \mathcal{O}_{ik}\mathcal{O}_{kj}$.

This is possible since for Majorana pairs we find $(m_i\, m_j) = (m_i\, m_k) \cdot (m_k\, m_j)$. We report only a 'slightly' higher weight as $\mathcal{O}_{ik}$ and $\mathcal{O}_{kj}$ have been introduced to localize their respective links in the first place. With the same argument we can take a walk over an arbitrary sequence of indices k_1, k_2, $\ldots$, k_l, where k_s and k_{s+1} are connected by an operator $\mathcal{O}_{k_s k_{s+1}}$, just to obtain the operator that links the first and the last mode k_1 and k_l

$$\mathcal{O}_{k_1 k_l} \propto \prod_{s=1}^{l} \mathcal{O}_{k_s k_{s+1}} \,. \tag{3.47}$$

This is the foundation for the Manhattan-distance property of all three mappings.

3.9.3.3 Verstraete-Cirac transform

Review Here we will review the Verstraete-Cirac transform starting with the original proposal [25], that, like the AQMs, can be regarded as manipulation of the Jordan-Wigner transform in which nonlocal strings are canceled with stabilizers. There, the auxiliary degrees of freedom that produce these stabilizers are added on the side of the model, where we find them in the form of Majorana modes. However, in the investigation of this mapping we found the consideration of the mapping as a quantum code more practical for a rigorous derivation of the stabilizers and outputs. This is why after a short motivation in the original language, we will describe the general concept of this mapping as a quantum code quite similar to the concept of the Auxiliary Qubit codes, which allows for the description of customized mappings such as a mapping with an odd number of rows or a qubit-economic version.

The idea of [25] is to extend the fermionic systems by doubling the number of modes, where the modes added are denoted by primed numbers from $1'$ to N'. For all indices k, k' does not denote another variable but is the primed version of the value of k. For the Jordan-Wigner transform, we need to impose the canonical order of $2N$ sites, and so we stagger primed and unprimed indices:
$1,\, 1',\, 2,\, 2',\, \ldots\ N,\, N'$. Adding those primed sites, we practically increase the length of Pauli strings, since all hopping terms on the original system hop over primed sites, even turning horizontal nearest-neighbor hoppings into next-nearest neighbor interactions.

$$\begin{aligned}(i<j): \qquad c_i^\dagger c_j + c_j^\dagger c_i \ &\hat{=}\ \frac{1}{2}\left[\bigotimes_{k=i+1}^{j-1} Z_k\right](X_i\otimes X_j + Y_i \otimes Y_j)\\ \mapsto\ \frac{1}{2}\left[\bigotimes_{k=i+1}^{j-1}(Z_k\otimes Z_{k'})\right]&(X_i\otimes Z_{i'}\otimes X_j + Y_i\otimes Z_{i'}\otimes Y_j)\ . \qquad (3.48)\end{aligned}$$

The hopping terms are thus made sensitive to the primed subsystem, and the original system is recovered if all primed modes are empty. In their original work, Verstraete and Cirac define a fermionic quantum code, that constrains the primed subsystem completely by means of majoranic stabilizers $(i\, m_{j'}\, \overline{m}_{k'})$ for certain pairs of modes j' and k'. These are translated to the qubit side by Jordan-Wigner transform $(i\, m_{j'}\, \overline{m}_{k'}) \,\hat{=}\, \mathcal{P}_{jk}$. While in the original proposal, the majoranic stabilizers $(i m_{j'} \overline{m}_{k'})$ are fixed as gap terms in the model Hamiltonian, it is suggested in [55] to prepare the entangled state by making syndrome measurements with the transformed stabilizers $\mathcal{P}_{jk}$.

Stabilizers like $(i m_{j'} \overline{m}_{k'})$ are useful to cancel nonlocal connections between j and k. Let us here assume that such a stabilizer is present, then the hopping between those modes can be modified by multiplication of the corresponding fermionic terms in the model Hamiltonians:

$$\begin{aligned}\left(c_j^\dagger c_k + c_k^\dagger c_j\right) i\, m_{j'}\, \overline{m}_{k'} \,\hat{=}\, &-\frac{1}{2}\, X_j \otimes X_{j'} \otimes Y_k \otimes Y_{k'} \\ &+ \frac{1}{2}\, Y_j \otimes X_{j'} \otimes X_k \otimes Y_{k'}\,. \end{aligned} \tag{3.49}$$

As one can see, the re-sized parity string has been canceled. Although all operators involved satisfy the correct (anti-)commutation relations, it is not possible to attribute the correct sign to all stabilizers and Hamiltonian terms without considering the code space. To do so, we now derive the quantum code version of the VCT, starting by the constructing the logical basis, that has to determine the adjustments to the Jordan-Wigner-transformed Hamiltonian terms. Although it was recently pointed out in [55], that keeping the stabilizers majoranic is unnecessary, we will stick to the original concept and merely add the freedom to 'flip' the stabilizer by introducing a sign

$$\mathcal{P}^{b_s}_{\alpha_s \beta_s} \,\hat{=}\, (-1)^{b_s}\, i\, m_{\alpha'_s} \overline{m}_{\beta'_s}\,, \tag{3.50}$$

where $\boldsymbol{\beta}$, $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_r) \in [N]^{\otimes r}$ and $\boldsymbol{b} = (b_1, b_2, \ldots, b_r) \in \mathbb{Z}_2^{\otimes r}$ are sequences that parameterize the mapping. A 'flipped' stabilizer would practically be implemented by requiring that syndrome measurements have the outcomes (-1), so a stabilizer $\mathcal{P}^{b_s}_{\alpha_s \beta_s}$ constrains the code space to $\langle \mathcal{P}^{0}_{\alpha_s \beta_s} \rangle = (-1)^{b_s}$. Instead of the primed and unprimed subspace to host indistinguishable fermions and being interleaved in the canonical order, we separate those modes (qubits) in an attempt to regard the primed subspace as the auxiliary register. The aux-register is not even required to have size N, instead a smaller number of auxiliary qubits can be chosen, $r \leq N$. Although separated into different registers, each auxiliary qubit is still affiliated with a data qubit, or rather their corresponding modes are. Our intention is to keep the previous notation and let the auxiliary register contain the primed labels. For that purpose, we introduce the set W as a r-sized subset of the mode numbers, $W \subseteq [N]$, such that the auxiliary register is comprised of qubits labeled $(\bigcup_{k \in W} k')$. In this way every data qubit $k \in W$ has an auxiliary qubit k' associated with it. Let us now characterize a general version of this mapping. We consider the $(\ell_1 \times \ell_2)$ block of data qubits and for every $s \in [r]$ connect the qubits α_s and β_s in a directed graph. For every qubit k that is a vertex of this graph, we add an auxiliary qubit k' somewhere, and the number k

becomes a member of W. Generalizing (3.50), the stabilizers of the qubit system are

$$\begin{aligned}\mathcal{P}^{b_s}_{\alpha_s\beta_s} &= (-1)^{b_s}\left(\bigotimes_{j=\alpha_s+1}^{\beta_s} Z_j\right)\otimes Y_{\alpha'_s}\otimes\left(\bigotimes_{\substack{k\in W\\ \alpha_s<k<\beta_s}} Z_{k'}\right)\otimes Y_{\beta'_s} \qquad \text{if} \quad \alpha_s<\beta_s\\ &= (-1)^{b_s}\left(\bigotimes_{j=\beta_s+1}^{\alpha_s} Z_j\right)\otimes X_{\beta'_s}\otimes\left(\bigotimes_{\substack{k\in W\\ \beta_s<k<\alpha_s}} Z_{k'}\right)\otimes X_{\alpha'_s} \qquad \text{if} \quad \alpha_s>\beta_s\,.\end{aligned} \tag{3.51}$$

Note that for the quantum code, that we intent to construct with the set $\{\mathcal{P}^{b_s}_{\alpha_s\beta_s}\}_{\boldsymbol{\alpha},\boldsymbol{\beta}}$ as stabilizer generators, certain conditions on $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are to be met. While these conditions are intrinsically fulfilled for the mappings in [25], we want to briefly spell them out for the sake of generality. The following conditions must be met by the directed graph on which $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are defined: (i) the graph must be composed of closed loops on the $(\ell_1 \times \ell_2)$-grid. (ii) The loops do not overlap in their vertices. (iii) The loops are uniformly directed, which means that within one loop no two edges point towards the same vertex.

Statement (i) is just a consequence of the fact that we need to constrain the auxiliary system completely. As the stabilizers (3.51) are associated with edges, we need to consider closed loops, otherwise degrees of freedom remain undetermined. We also need to make sure that all stabilizes commute and so, considering (3.50), we find that every vertex can host one incident and one outbound edge. This, together with statement (i), explains statements (ii) and (iii). An example of such a mapping, for which all three statements hold, is depicted in Figure 3.12(a), where we consider two loops in counter-clockwise directions. While in (a), we eliminate some arbitrary nonlocal connections, Figure 3.12(b) exhibits the original proposal, where the stabilizer implement the vertical connections. Of course we need to involve a few horizontal connections in order to comply with statement (i). As loops cannot be closed in it, the original proposal deals with an odd number ℓ_1 in ignoring the last column. Alternatively, we suggest that one could just create loops between vertically adjacent modes in that last column, like it is done in the right-most loop in panel (a). It is of course only possible to stabilize roughly half of all vertical connections in this way, i.e. all even or all odd pairs. Assuming an underlying S-pattern of the canonical ordering, nothing else would be required, since half of the links are local anyways. The original proposal yields a decent mapping already, as we can shorten vertical hoppings along the last column by multiplications stabilizers of the second-to-last column. In fact, the idea that not every column needs to have their own auxiliary qubits is the foundation for qubit-conserving versions of the VCT, as is shown in Figure 3.12(c). Note that in order to comply with the three statements, the periodicity $\mathcal{I}$ has to be chosen such that $(\ell_1 - 1)/\mathcal{I}$ is an odd number, the size of the auxiliary register subsequently becomes $r = \ell_2 + (\ell_1 - 1)\ell_2/\mathcal{I}$. Note that a loop of one vertex is counterproductive, resulting in a stabilizer $\mathcal{P}^b_{jj} = (-1)^{1+b} Z_{j'}$. This only fixes the parity of the auxiliary qubit, which renders it redundant since it is not entangled with the rest of the system. Not just that, it blocks the mode from being part in another loop.

Let us now take a look to the basis of the extended system. As before, the N original modes describing the fermionic Fock space shall make up the data qubit register and the primed auxiliary qubits be in the register aux $= (\bigcup_{k \in W} k')$. An ansatz for a logical basis stabilized by all $\{\mathcal{P}^{b_s}_{\alpha_s \beta_s}\}$ is

$$|\boldsymbol{\omega}\rangle_{\text{dat}} \mapsto \propto \left(\sum_{\boldsymbol{\mu} \in \mathbb{Z}_2^{\otimes r}} \prod_{s=1}^{r} \left[\mathcal{P}^{b_s}_{\alpha_s \beta_s} \right]^{\mu_s} \right) |\boldsymbol{\omega}\rangle_{\text{dat}} \otimes |\boldsymbol{\chi}\rangle_{\text{aux}} , \tag{3.52}$$

where $|\boldsymbol{\chi}\rangle_{\text{aux}} = (\bigotimes_{k \in W} |\chi_k\rangle_{k'})$ is a product state on the auxiliary register that can be chosen inside a certain range of parity constraints, which we now want to explain.

These parity constraints are related to a certain freedom in the characterization of the mapping. We have not determined $\boldsymbol{b}$ yet, as up to now the only restrictions we had were on the choice of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. In order to understand the role of $\boldsymbol{b}$, let us for a moment assume that the graph spanned by $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ is only one loop, which means that $\beta_s = \alpha_{s+1}$ and $\beta_r = \alpha_1$. No matter the number of loops, the sum in the basis (3.52) will always contain the product of all stabilizers around a closed loop, here it is $(\prod_{s=1}^{r} \mathcal{P}^{b_s}_{\alpha_s \beta_s})$, met by the summand for which $\boldsymbol{\mu} = (1)^{\otimes r}$. In fact, half of the terms in the sum will differ from the other half only by these operators: (having omitted the normalization factor for that reason) it is alright for some stabilizers to be linearly dependent, as long as they stabilize $|\boldsymbol{\omega}\rangle_{\text{dat}} \otimes |\boldsymbol{\chi}\rangle_{\text{aux}}$. Since we are stabilizing a loop, we find by (3.50),

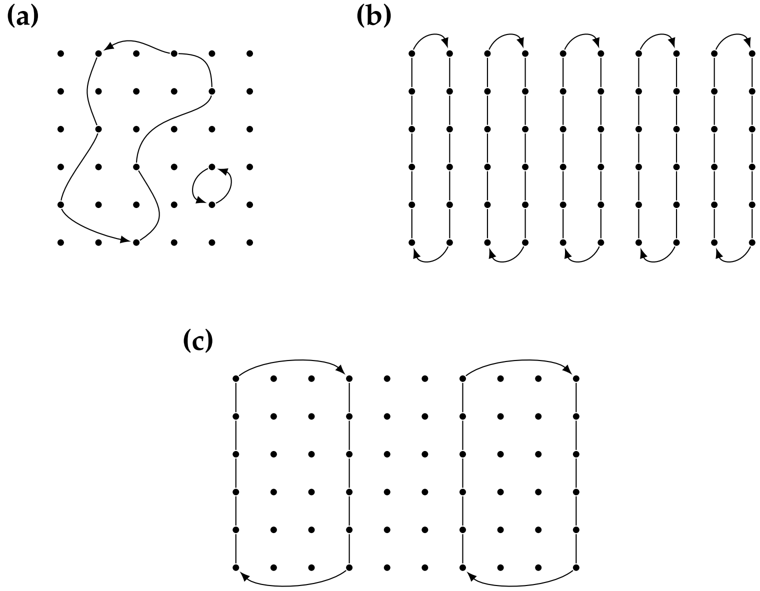


Figure 3.12. Verstraete-Cirac transform. **(a)** An arbitrary mapping showcasing the constraints on the VCT code space. The black dots correspond to data qubits. Directed loops of operators $\mathcal{P}^b_{jk}$ are drawn into this grid, where the direction of one loop is indicated by arrows. With 9 vertices involved, we entangle 9 auxiliary qubits to that system. **(b)** Graph of the original proposal [25]. **(c)** One possibility for a qubit-economic version of the VCT.

that

$$\prod_{s=1}^{r} \mathcal{P}^{b_s}_{\alpha_s \beta_s} = (-1)^{1+\sum_{k=1}^{r} b_k} \bigotimes_{j \in W} Z_{j'} \,. \tag{3.53}$$

Since (3.53) acts only on $|\chi\rangle_{\text{aux}}$, it becomes apparent that $\boldsymbol{b}$ determines the parity of all auxiliary qubits associated with the loops in the mapping. According to the choice of $\boldsymbol{b}$, we now need to pick a state $|\chi\rangle_{\text{aux}}$ that meets all parity constraints (3.53). Since we in general have more than one loop in our mapping, we need to fix the parity on several distinct subsets of $|\chi\rangle_{\text{aux}}$. For instance if we pick the parity of every loop to be even, we can choose $|\chi\rangle_{\text{aux}} = |0^r\rangle_{\text{aux}}$.

We lastly show that Z-strings on the primed qubits come naturally as adjustments to Hamiltonian terms h_{dat}, together with minus signs from the loop parity constraints. The data-qubit substring of the stabilizers (3.51) is purely a Z-string, so we do not need to adjust a string $h_{\text{dat}} \in \{\mathbb{I},\, Z\}^{\otimes N}$. This means that it is sufficient to consider the changes to be made to a string $(\bigotimes_{j=1}^{k-1} Z_j) \otimes X_k$, in order to describe all fermionic operators $c_k^{(\dagger)}$. This string anticommutes with all stabilizers, that have data qubit substrings $(\bigotimes_{j=s}^{t} Z_j)$, where $s \leq k$. These stabilizers, $\mathcal{P}^b_{(s-1)t}$ or $\mathcal{P}^b_{t(s-1)}$, act on the aux-register as

$$(-1)^b \, Y_{(s-1)'} \otimes \left(\bigotimes_{\substack{j \in W \\ t<j<(s-1)}} Z_{j'} \right) \otimes Y_{t'}$$

$$\text{or} \qquad (-1)^b \, X_{(s-1)'} \otimes \left(\bigotimes_{\substack{j \in W \\ t<j<(s-1)}} Z_{j'} \right) \otimes X_{t'} \,, \tag{3.54}$$

which means they change the parity of the subsystem that is spanned by all auxiliary qubits with the labels j', where $j \leq (k-1)$ and $j \in W$. The total parity of all auxiliary qubits is however constant i.e. it does not change with the multiplication of either stabilizer. The total parity is predetermined by $|\boldsymbol{\chi}\rangle_{\text{aux}}$ and the action of a Majorana-pair operator conserves it.

If we now multiply $(\bigotimes_{j=1}^{k-1} Z_j) \otimes X_k$ to a basis element (3.52), we can determine whether it anticommutes with an even or odd number of stabilizers as we move it to the right until it reaches $|\boldsymbol{\omega}\rangle_{\text{dat}} \otimes |\boldsymbol{\chi}\rangle_{\text{aux}}$: it anticommutes with an odd number of stabilizers if the parity of the subsystem, spanned by all auxiliary qubits with labels at most as large as $(k-1)'$, is changed. We therefore extract the parity of said subsystem by the operator $(\bigotimes_{j \in W < k} Z_{j'})$ and add a minus sign in case $(\bigotimes_{j \in W<k} Z_{j'}) \, |\boldsymbol{\chi}\rangle_{\text{aux}} = (-1) \, |\boldsymbol{\chi}\rangle_{\text{aux}}$. We hence find

$$\left(\bigotimes_{i=1}^{k-1} Z_i \right) \otimes X_k \;\mapsto\; \pm \left(\bigotimes_{j=1}^{k-1} Z_j \right) \otimes X_k \otimes \left(\bigotimes_{j \in W<k} Z_{j'} \right) \tag{3.55}$$

where the sign is determined by $|\boldsymbol{\chi}\rangle$. When we consider the planar code of the original proposal, we find that string has become

$$\pm \left[\bigotimes_{j=1}^{k-1} (Z_j \otimes Z_{j'}) \right] \otimes X_k \tag{3.56}$$

which is the expected string with perhaps a minus sign, depending on whether we have flipped any stabilizers. Note however that the loop parity constraints have to be fulfilled somewhere, either by minus signs in the logical operators or by flipping stabilizers.

Adaption to the layout & Manhattan-distance property We here adapt the Verstraete-Cirac transform to the square lattice connectivity, such that it has the Manhattan-distance property. In doing so, we will not stray too far from the original proposal, that is built upon the connectivity graph in Figure 3.12(b). The layout is roughly motivated by an S-pattern of the qubits ordered $1\ 1'\ 2\ 2'\ \dots N\ N'$. For reasons that become clear later, we need the rows to be connected vertically by the auxiliary qubits, which leads us to shift every second row in order to align the primed qubits. The vertical connections are also placed along the windings of the S-pattern, resulting in a graph that can be studied in Figure 3.13(a). For the initialization of a state, stabilizers that are horizontally adjacent are multiplied pairwise. We fully constrain the auxiliary systems by those localized stabilizers, plus the stabilizers that are local already: the ones along the windings and the horizontal connections in the first and ℓ_2-th row. The stabilizer tiling to the layout of Figure 3.13(a) is presented in panel (b) of the same figure. As already remarked in [25], the analogy of the stabilizer tilings of this code and the rotated surface code [17] comes to mind easily. The tiles of the VCT are identical to the surface code on the primed qubits, but the stabilizers contain some additional Z-strings on the data qubits. Also, not all of the stabilizers might have the same sign according to b in the definition $\mathcal{P}^b_{jk}$. Curiously, only the first qubit of the data register is not entangled with the auxiliary system in any way.

Using the interpretations of the stabilizers (3.50), we can define $\mathcal{O}_{jk} \propto (-1)^b\ \mathcal{P}^b_{jk} Z_{k'}$ and obtain arbitrary long-range vertical connections over the sequence of vertically aligned stabilizers $\mathcal{P}^{b_s}_{k_s k_{s+1}}$, where $\boldsymbol{k} \in [N]^{\otimes l}$ and $\boldsymbol{b} \in \mathbb{Z}_2^{\otimes l}$, via (3.47):

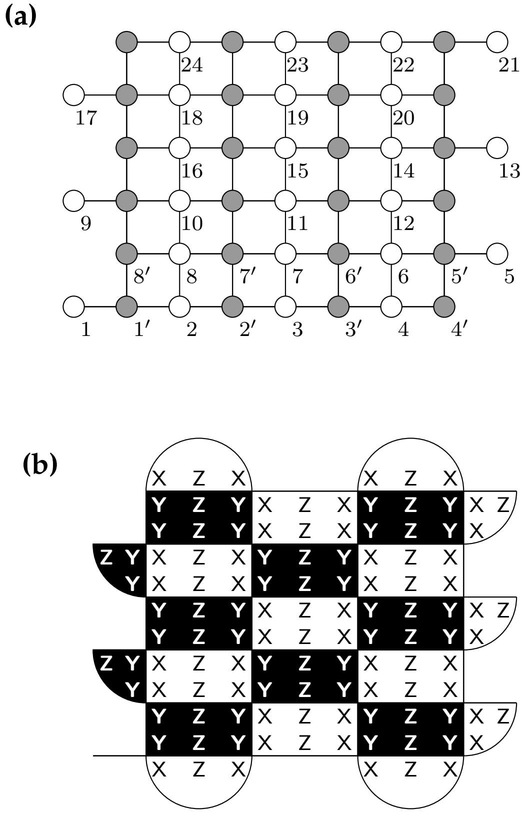


Figure 3.13. VCT as a planar code. **(a)** Connectivity graph, in which we alternate data (white) and auxiliary qubits (gray), but shift every second row such that the auxiliary qubits align vertically. The labeling of the qubits follows an S-pattern. **(b)** Stabilizers of the VCT for a graph as in Figure 3.12(b), the original proposal. We here give the connectivity graph a two-coloring of the stabilizer plaquettes, where the Pauli operators, that make up each stabilizer, are denoted by letters inside the plaquettes close to where their corresponding qubits are. Note that we have not indicated the signs that each stabilizer possibly has attached to it.

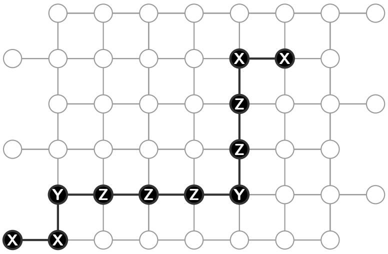


Figure 3.14. Simulating the term $(im_{20}\,\overline{m}_1)$ via the VCT, where we have arbitrarily deformed the string by the multiplication of stabilizers.

$$\prod_{t=1}^{l-1} \mathcal{P}^{b_t}_{k_t k_{t+1}} = \mathcal{P}^{a}_{k_1 k_l} \bigotimes_{u=2}^{l-1} Z_{k'_u}\,, \tag{3.57}$$

where $a = (\sum_{s=1}^{l} b_s)$. Equation (3.57) means that the multiplication of these vertical stabilizers yields a nonlocal connection $\mathcal{P}^{a}_{k_1 k_l}$, which (is not a stabilizer and) is missing the operators $Z_{k'_u}$ for $1 < u < l$. The absence of these Z-operators does not cancel them in Pauli strings originating from fermionic terms like $c_i^\dagger c_j$, where $i \leq k_1 < k_l \leq j$. These operators subsequently serve as connection between the qubits labeled k'_1 and k'_l, as the qubits are vertically aligned by our layout. With this building block we can multiply various stabilizers and so connect the qubits i and j via different paths but with the same number of gates. In Figure 3.14, we present an example of such a term.

3.9.3.4 Superfast Simulation

Review We here review the original proposal of the Bravyi-Kitaev Superfast simulation, [26], which includes the transform of the operators and the structure of the stabilizers.

In contrast to the other mappings, the Superfast simulation is not defined to transform fermionic operators, but pairs of Majoranas. Thus the BKSF only allows us to conveniently consider Hamiltonians that conserve the fermionic parity i.e. are comprised of operator pairs $c_j c_k$, $c_j^\dagger c_k^\dagger$ and $c_j^\dagger c_k$. By the relations (3.43), these Hamiltonians can then be expressed using only the operators

$$\mathcal{A}_{jk} \hat{=} -i\, m_j\, m_k\,, \tag{3.58}$$

$$\mathcal{B}_{k} \hat{=} -i\, m_k\, \overline{m}_k\,, \tag{3.59}$$

where $\mathcal{A}_{jk}$ and $\mathcal{B}_k$ are some Pauli strings. Using these operators, fermionic Hamiltonians can be transformed via

$$c_j c_k \hat{=} \frac{i}{4}\left(\mathcal{A}_{jk} - \mathcal{A}_{jk}\mathcal{B}_k + \mathcal{B}_j\mathcal{A}_{jk} - \mathcal{B}_j\mathcal{A}_{jk}\mathcal{B}_k\right)\,, \tag{3.60}$$

$$c_j^\dagger c_k^\dagger \hat{=} \frac{i}{4}\left(\mathcal{A}_{jk} + \mathcal{A}_{jk}\mathcal{B}_k - \mathcal{B}_j\mathcal{A}_{jk} - \mathcal{B}_j\mathcal{A}_{jk}\mathcal{B}_k\right)\,, \tag{3.61}$$

$$c_j^\dagger c_k \hat{=} \frac{i}{4}\left(\mathcal{A}_{jk} - \mathcal{A}_{jk}\mathcal{B}_k - \mathcal{B}_j\mathcal{A}_{jk} + \mathcal{B}_j\mathcal{A}_{jk}\mathcal{B}_k\right)\,. \tag{3.62}$$

The BKSF is furthermore not based on the Jordan-Wigner transform, so $\mathcal{A}_{jk}$ and $\mathcal{B}_k$ are not going to be obtained by transforming the right-hand side of (3.58) and (3.59) under (3.44). Instead, the $\mathcal{A}$- and $\mathcal{B}$-operators will be defined on a unique qubit layout, that we now introduce.
The Hamiltonian that we want to simulate describes a certain graph of pairwise interactions between modes, for example there is an edge between vertices j, k when it contains at least one of the term (3.60)-(3.62). The qubit connectivity graph of the Superfast simulation is then the line graph of this Hamiltonian graph. Here the operators $\mathcal{A}_{jk}$ are associated with edges in the Hamiltonian graph, i.e. interactions of the Hamiltonian, and the operators $\mathcal{B}_k$ are associated with vertices, i.e. fermionic modes. Let E be the set of undirected edges of the Hamiltonian graph, and ε_{jk} a number associated to the index pair jk, that yields zero if $jk \notin E$. By means of ε_{jk} a direction on the graph is fixed by imposing that if $jk \in E$, then $\varepsilon_{jk} = 1$, in case the edge is directed from $j \mapsto k$, and $\varepsilon_{jk} = -1$ when the direction is opposite. With that construction, we will take into account that $\mathcal{A}_{jk} = -\mathcal{A}_{kj}$, which is straightforward to see from (3.58). Also, on every vertex k, we need to impose an ordering of the edges connected to it. To that end Bravyi and Kitaev introduce the symbolic operator $\underset{k}{<}$, such that two different edges $jk,\ lk \in E,\ j \neq l$ on vertex k are ordered by a relation like $jk \underset{k}{<} lk$. As we place the qubits on the edges of that graph, both jk and kj shall be identifiers for the same qubit (given $\varepsilon_{jk} \neq 0$). In the original BKSF, the number of qubits equals the number of edges in the graph, so the qubit requirements do not depend on the system size,

but on the size of the Hamiltonian. The operators $\mathcal{A}_{jk}$ and $\mathcal{B}_k$ are defined by

$$\mathcal{B}_k = \bigotimes_{a:\, ak \in E} Z_{ak}\,, \tag{3.63}$$

$$\mathcal{A}_{jk} = \varepsilon_{jk} X_{jk} \left(\bigotimes_{b:\, bk \underset{k}{<} jk} Z_{bk} \right) \left(\bigotimes_{c:\, jc \underset{j}{<} jk} Z_{jc} \right) . \tag{3.64}$$

As shown in [26], these operators fulfill all algebraic relations that we would expect from representations of (3.58) and (3.59) but one. As it is now, the mapping would allow a Majorana to unphysically interact with itself via hopping terms around a closed loop. For a length-l sequence $a_1,\, a_2,\, a_3,\, \ldots, a_l$, that describes a closed loop along edges, i.e. $a_j a_{j+1} \in E$ and $a_1 = a_l$, we must impose that

$$(i)^l \prod_{j=1}^{l-1} \mathcal{A}_{a_j a_{j+1}} \tag{3.65}$$

is a stabilizer of the system. As not all closed loops are linearly independent, one needs to stabilize only the smallest closed loops of the system.

Adaption to the layout & Manhattan-distance property We now adapt the Superfast simulation to the square lattice layout and give it the Manhattan-distance property. As we are interested in simulating more than square lattice Hamiltonians, we are going to depart a bit from the original concept of the qubit connectivity being related to the Hamiltonian.

Instead, we will show that we can adapt the mapping adequately by pretending that the Hamiltonian graph is a square lattice. On this lattice, modes that are actually subject to hopping interactions in the Hamiltonian, should be locally close. Such a lattice of modes is shown in Figure 3.15(a), where the direction of every edge is indicated. As the direction of every edge jk only determines the factor $\varepsilon_{jk} \in \{+1,\, -1\}$ in (3.63), it has not much influence on the transformation. We will see later that the choice of the order of the edges on every mode is way more relevant for the strings that such a mapping produces. In 3.15(a), we have already outlined the tiling of the line graph, to which we now switch. The resulting qubit connectivity graph can be seen in Figure 3.15(b), where the

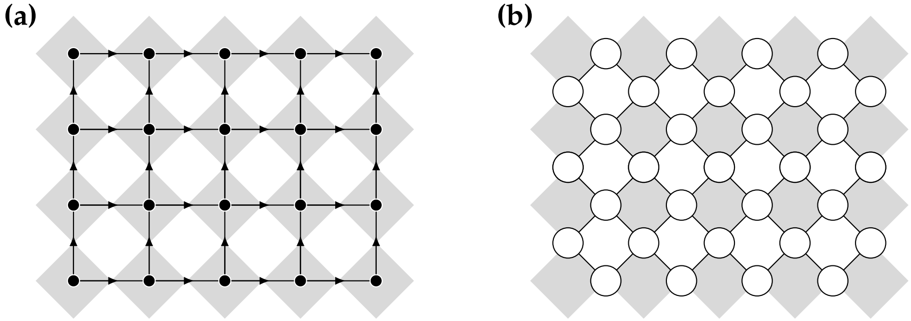


Figure 3.15. Connectivity graphs for Superfast simulation in limited connectivity. **(a)** Hamiltonian graph: all vertices correspond to fermionic modes, and in the original setting all edges would indicate the presence of hopping terms between the two modes in the Hamiltonian. We have displayed the direction of every edge in this graph. **(b)** Qubit connectivity graph: a qubit is placed on each vertex of this rotated square lattice. The underlying checkerboard pattern indicates which qubits are associated with which fermionic modes. Each dark plaquette is associated with an index k, such that the qubits on each of its corners have indices $jk \in E$.

plaquettes enclosing a fermionic mode are darkened. Starting from a general set of $\ell_1 \times \ell_2$ modes, we have now ended up with a rotated patch of the square lattice that has $2\ell_1\ell_2 - (\ell_1 + \ell_2)$ qubits on it. The number of white plaquettes, that are enclosed in the graph, describes the number of smallest possible loops, which means it is the total number of linearly independent stabilizers. We have $(\ell_1 - 1)(\ell_2 - 1)$ of those white plaquettes, which means the system has $2^{\ell_1\ell_2 - 1}$ degrees of freedom left: since we have mapped only pairs of operators (3.60)-(3.62), we are now seemingly stuck in the subspace with an even number of fermions. This situation is however not terminal: we can simulate the odd-parity subspace separately as well as the entire Fock space. Let us further illuminate this issue by considering the logical basis of the even-parity subspace first. For that purpose we pick a set $\{S^i\}_i$ of $(\ell_1 - 1)(\ell_2 - 1)$ linearly independent stabilizers from (3.65). The set fully constrains the system. Automatically, all stabilizers S^i are orthogonal in the computational basis, such that the fermionic vacuum state is encoded as

$$|\Theta\rangle \hat{=} \left[\prod_i \frac{1}{\sqrt{2}} \left(\mathbb{I} + S^i\right)\right] |0^n\rangle \ . \tag{3.66}$$

We can then apply operators $\mathcal{A}_{jk}$ and $\mathcal{B}_j$ in order to prepare other states with an even particle number. While the $\mathcal{A}_{jk}$ are different for every ordering, the operators $\mathcal{B}_k$, are independent of it: an operator $\mathcal{B}_k$ is the string of Z-operators around the shaded plaquette associated with mode k. If this plaquette is in the interior of the lattice in Figure 3.15(b), the string has weight four, three if it is on the boundary edge, and two if in a corner. The one feature that the operators $\mathcal{A}_{jk}$ have in common for every ordering, is that they include an X-operator on the qubit (jk). Apart from the administration of some minus signs, the $\mathcal{A}_{jk}$ has generally the effect to flip qubit (jk) in the all-zero state $|0^n\rangle$ of (3.66). Comparing the encoded operators (3.58) and (3.59) to the toy picture of the $\mathcal{A}$ and $\mathcal{B}$ operators we have just suggested, we find that a qubit configuration $|\boldsymbol{\xi}\rangle = (\bigotimes_{jk\in E} |\xi_{jk}\rangle_{jk})$, with all $\xi_{jk} \in \mathbb{Z}_2$, has the following correspondence to a fermionic quantum state:

$$\left[\prod_i \frac{1}{\sqrt{2}} \left(\mathbb{I} + S^i\right)\right] |\boldsymbol{\xi}\rangle \;\; \hat{\propto} \;\; \left[\prod_{j=1}^{N} \left(c_j^\dagger\right)^{\sum_{i:\,(ij)\in E} \;\xi_{ij} \mod 2}\right] |\Theta\rangle \; . \qquad (3.67)$$

Note that (as denoted by $\hat{\propto}$) we have not kept track of any minus signs in (3.67). The relation is however sufficient to show that a fermionic mode k is occupied, if an odd number of qubits around the plaquette k are in $|1\rangle$. The product of the stabilizers $\prod_i \frac{1}{\sqrt{2}} \left(\mathbb{I} + S^i\right)$ mixes all possible configurations that conserve the common parity of qubits around a shaded plaquette (as the stabilizers need to commute with $\mathcal{B}_k$, a logical operator), and so the fermionic occupations are conserved as well. In order to prepare a pure fermionic state different from the vacuum, we need to consider a qubit configuration $|\boldsymbol{\xi}\rangle$, in which we flip strings of adjacent qubits in order to create fermions on the plaquettes at their ends, see Figure 3.16.

So far, we still have not left the even-parity subspace, but we might have systems to solve that are populated by odd numbers of fermions. In [57], it is suggested to add another mode to the system that is however not coupled to any other term in the Hamiltonian. From the original concept of the BKSF it is however not clear how this mode is brought into the system, since all qubits correspond to couplings of modes in the Hamiltonian, which here do not exist. Let us suggest to couple this mode to exactly one other, without ever using the $\mathcal{A}$-operator of this link in the Hamiltonian. For state preparation we however can have strings that end at that outer plaquette, creating a mode that does not play a role and

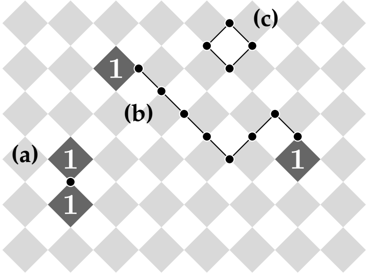


Figure 3.16. State preparation in the Superfast simulation. Black dots are flipped qubits and plaquettes with an odd number of flipped qubits are marked with **1**, as a fermion is created on the corresponding mode. **(a)** Flipping a qubit with label (jk) creates fermions on the adjacent modes j and k. **(b)** X-strings (here emphasized by linking the qubits) create nonlocal pairs of fermions, as long as we ensure to flip always an even number of qubits on each plaquette, which means winding around white plaquettes when the string has to change direction. **(c)** Flips like this result from stabilizers, and do not excite fermions, as on all dark plaquettes an even number of qubits is flipped.

so effectively increase the degrees of freedom to 2^N, modeling the entire Fock space. The cost of this increase is the overhead of one qubit. Alternatively there is a way to only map the odd-parity subspace without using additional quantum resources: the idea is to consider the plaquette k as being switched to 'filled', such that the configuration on the right-hand side of (3.66) does not correspond to the vacuum state (which is in the even-parity subspace), but to the state $c_k^\dagger \left|\Theta\right\rangle$. Flipping the qubit (jk) will lead to the fermion on k being annihilated and re-created on j, a string of flips that ends at k will in general move the fermion to the other end. We therefore make the replacement $\sum_i \xi_{ik} \mapsto (1 + \sum_i \xi_{ik})$ in the exponent of the mode-k creation operator $c_k^\dagger$ on the right-hand side of (3.67). In order to account for the switched occupation, we also need to update $\mathcal{B}_k \mapsto (-1)\mathcal{B}_k$ and add minus signs to some $\mathcal{A}$-operators.

After having established an abstract idea of BKSF on the square lattice, we will now consider different versions of this mapping as we delve into detail. As mentioned before, the stabilizers of this mapping roughly flip qubits around white plaquettes. Due to (3.65), their exact structure is determined by the operators $\mathcal{A}_{jk}$, which on the other hand depend on the ordering of edges on every vertex in the Hamiltonian graph, Figure 3.15(a). In the qubit graph, this means that with every shaded plaquette we associate numbers with the qubits on its edges. The decision for an ordering has to be made consciously, as it influences the weight of strings simulating long-range hoppings. For now let us consider two different versions of this mapping in Table 3.7. For each version we assume that the ordering on every dark plaquette (leaving out missing vertices at the boundaries) is the same. From (3.63), we therefore just need to differentiate between vertical and horizontal version of the operators $\mathcal{A}_{jk}$, i.e. considering the directions of the edges, we need to separate the cases where (1) the plaquette k is the right neighbor of the plaquette j and (2) where the plaquette j is below k. In the Table 3.7, we sketch these operators, along with the stabilizers that follow from the multiplication of four of those operators to describe a closed loop around a white plaquette. The first version is the one already considered in [29], and second one is related to the mapping in [54].

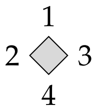
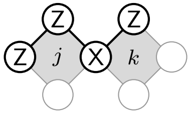
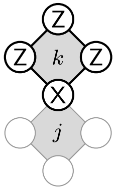
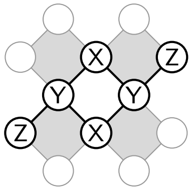
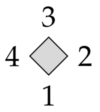
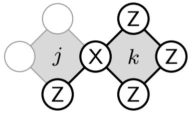
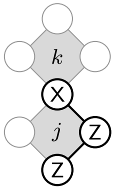
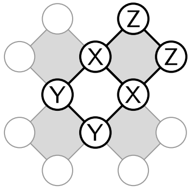
Ordering	$\mathcal{A}_{jk}$ (horizontal)	$\mathcal{A}_{jk}$ (vertical)	Stabilizer
			
			

Table 3.7. Different versions of BKSF. The ordering of the edges on each vertex is displayed as well as the operators this ordering entails: horizontal and vertical edge operators $\mathcal{A}_{jk}$ and the stabilizers (signs are omitted). The upper version is the one used in [29], while the lower one is related to the mapping in [54].

We can now describe fermion-operator-pairs via Table 3.7 with (3.60)-(3.62). The latter equations hold for operators $\mathcal{A}_{jk}$ of every link, whereas the table only provides us with operators in which j and k are adjacent plaquettes. We will now cease to pretend that the Hamiltonian is just composed of nearest-neighbor interactions, and derive nonlocal operators $\mathcal{A}_{jk}$. By (3.58) we set $\mathcal{A}_{jk} \propto \mathcal{O}_{jk}$ and using (3.47) we find

$$\mathcal{A}_{k_1 k_l} = (i)^{l-1} \prod_{s=1}^{l-1} \mathcal{A}_{k_s k_{s+1}} \tag{3.68}$$

for any sequence $k_1, k_2, \ldots, k_l$, where for all $s \in [l-1]$: $k_s k_{s+1} \in E$. This means we can multiply several of the nearest-neighbor $\mathcal{A}$-operators from Table 3.7. The choice of the ordering turns out to be crucial, as for various orderings, the resulting mapping is not a good one according to the criteria of Section 3.3. The first mapping in Table 3.7 for instance does not produce a continuous Pauli string (3.68) when making a chain of several horizontal $\mathcal{A}_{jk}$. For a vertical chain, we have a maximal operator weight. The second mapping on the other hand is better behaved: horizontal and vertical $\mathcal{A}$-operators are connected and their weight is minimal. In Figure 3.17, we present an example of the simulation of the Pauli string $(-i\mathcal{B}_{k_1}\mathcal{A}_{k_1 k_l})$, where $\mathcal{A}_{k_1 k_l}$ is nonlocal as in (3.68), with $l = 13$. The string here extends on a zig zag line along the edges of the plaquettes involved, $\{k_s\}_s$ connecting the plaquettes k_1 and k_{13}. The weight of this string can perhaps be optimized in cutting more corners like at plaquette k_5. In any case, we have adapted the BKSF as a two-dimensional fermion-to-qubit mapping on the square lattice.

3.9.3.5 Fermi-Hubbard model

In this section we test the proposed square lattice implementations of the Superfast simulation and the Verstraete-Cirac transform on the Fermi-Hubbard model.

For both mappings, we have to decide where to place spin-up and -down modes of the same spatial site. On the one hand should the qubits representing these modes be locally close, perhaps even horizontally or vertically adjacent, but on the other hand they will increase the weight of the strings simulating hopping terms, as they are 'in the way'. For the BKSF, it is almost inconsequential whether the spin pairs are vertically or horizontally stacked, so we decide for the latter. For the VCT, the situation is

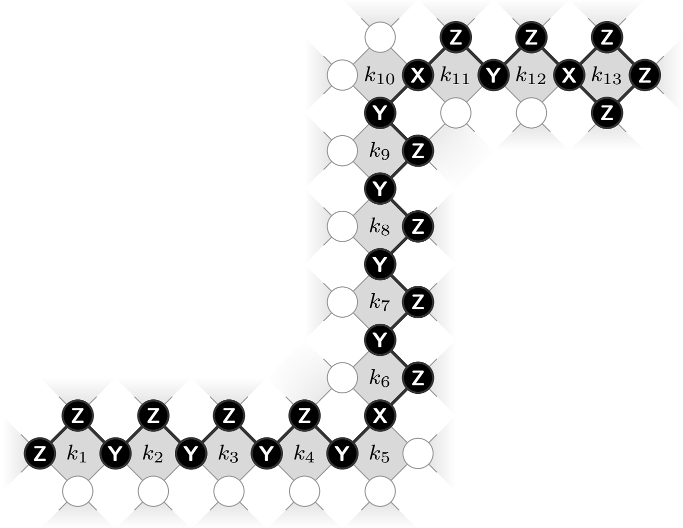


Figure 3.17. Superfast simulation of a hopping operator in the between modes k_1 and k_{13}, coupling the respective shaded plaquettes in a string of length scaling with their Manhattan distance, where the path taken is defined by the locally connected chain of modes k_2 to k_{12}. The string simulated is $(-i\mathcal{B}_{k_1}\mathcal{A}_{k_1 k_{13}})$, which in Jordan-Wigner transform would be $h_{\text{dat}} = (X_{k_1} \otimes Z_{k_1+1} \otimes \cdots \otimes Z_{k_{13}-1} \otimes X_{k_{13}})$. The plaquettes $(k_1, \ldots, k_{13})$ are labeled on this lattice.

different as it produces shorter hopping strings in the vertical direction, which leads us to make the spin pairs vertical neighbors on the grid. In order to do that, we need to compensate for the shift that has emerged aligning the primed qubits: in Figure 3.13(a), qubit 4 is for instance below qubit 6, not qubit 5. Without this shift, there would be additional costs for horizontal or vertical hoppings, but with the shift, additional costs emerge for the Hubbard terms. As a fix, we simulate the model with ℓ_2 additional modes, that remain empty. The qubits corresponding to those modes are the ones at the horizontal perimeter of the qubit lattice, i.e. the qubits labeled 1, 5, 9, 13, 17 and 21 in Figure 3.13(a). Those data qubits, fixed to $|0\rangle$, can as well be removed, but their primed counterparts must remain and be part of the code. The spin-partners can now be placed vertically adjacent on the grid. The Hubbard model with $L \times L$ spatial sites is thus simulated with $4L^2 + 2L$ qubits in the VCT, and with $4L^2 - 3L$ qubits in the BKSF. The resulting Pauli strings can be found in Table 3.8.

3.10 Notations

[...]	The set of integers from 1 to the argument.
$(i,\, j)$	Spacial coordinates replacing qubit labels in Section 3.5. To distinguish data from auxiliary qubits, the latter are given half-integer coordinates.
$\hat{=}$	Sign signifying the equivalence between fermionic operators/states to qubit counterparts according to some fermion-to-qubit mapping.
$A,\, A^{-1}$	The two binary $(N \times N)$ matrices defining linear fermion-to-qubit mappings, see Section 2.3 in the first chapter.
AQM	Auxiliary Qubit Mapping [43].
aux	The auxiliary qubit register, which comprises the integer labels from $(N + 1)$ to $(N + r)$.
BKSF	Superfast simulation of fermions on a graph, also known as Superfast Encoding, by Bravyi and Kitaev [26]
$c_j^\dagger, c_j$	Fermionic creation and annihilation operators on mode j.

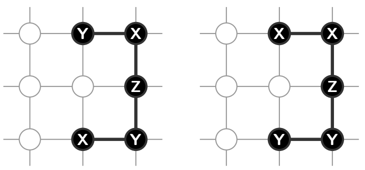
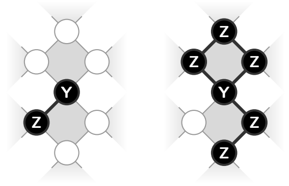
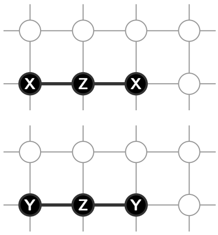
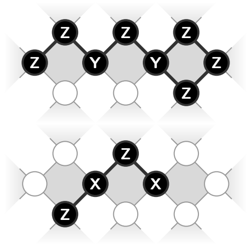
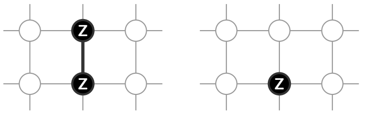
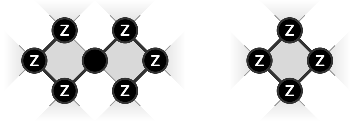
Verstraete-Cirac transform	Superfast simulation
Vertical hoppings	
	
Horizontal hoppings	
	
Hubbard interactions	
	

Table 3.8. Transforming terms of the Hubbard model according to the Verstraete-Cirac and Superfast simulation mapping. For the hoppings, we consider the real hopping terms, i.e. transforms of $(im_j\,\overline{m}_k)$ and $(im_k\,\overline{m}_j)$ for $j < k$. Note that for the Verstraete-Cirac transform, the vertical hopping terms are different for even/odd rows and columns. Here the south east qubit is in an even column and odd row. The qubit marked, but not labeled with X, Y or Z, is skipped.

$\textsc{CNot}(i \rightarrow j)$ $|0\rangle\langle 0|_i + |1\rangle\langle 1|_i \otimes X_j$. The Controlled-Not gate, where i is the control and j is the target qubit.

dat The data register labels, which comprises all integers from 1 to N.

$F(j)$ The flip set of mode j, see (2.12).

h_{dat}, $\widetilde{h}_{\text{aux dat}}$ N-qubit Pauli strings and their logical equivalents on $N+r$ qubits. Note that $\widetilde{h}_{\text{aux dat}}$ and $(h_{\text{dat}} \otimes \kappa^h_{\text{aux}})$ are equivalent and identical by the multiplication of stabilizers.

H_{dat}, $\widetilde{H}_{\text{aux dat}}$ An N-qubit physical Hamiltonian (3.3), as for instance obtained by Jordan-Wigner transform, and its $(N+r)$-qubit logical equivalent (3.7).

H_j $\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$. The Hadamard gate on qubit j.

$\mathbb{I}$ The identity matrix or operation in various spaces.

$\mathcal{I}$ A parameter determining the periodicity in the sparse AQM, see Table 3.1.

κ^h_{aux} A Pauli string on the auxiliary register. Adjustments made to the physical operator h_{dat}, see (3.6).

ℓ_1, ℓ_2 Parameters of the lattice. $\ell_1 \times \ell_2$ is the dimension of the fermionic lattice depicted in Figure 3.1(a).

L A lattice size. $2L \times L$ is the dimension of the Fermi-Hubbard-lattice in Section 3.6.

m_j, $\overline{m}_j$ The two types of Majorana operators associated with the same mode j, see (3.42)-(3.44).

n $N + r$. The number of qubits.

N $\ell_1 \times \ell_2$. The number of fermionic modes.

$P(j)$ The parity set of mode j, see (2.12).

p^i_{dat} The part of the stabilizers $(p^i_{\text{dat}} \otimes \sigma^i_{N+i})$ that is supported on the data register, see (3.5).

r The number of auxiliary qubits.

R A binary $(N \times N)$ matrix, in which the lower triangle is filled with ones, see see (2.13).

σ^i_{N+i} The part of the stabilizers $(p^i_{\text{dat}} \otimes \sigma^i_{N+i})$, that is supported on the auxiliary register, see (3.5).

$U(j)$	The update set of mode j, see (2.12).
$V_{\text{aux dat}}$	The unitary initializing the code space, see (3.5).
VCT	Verstraete-Cirac transform, also known as Auxiliary fermion mapping, [25] .

3.11 Further work

Generalized Superfast Encoding

Not long after we finished [43], the Superfast simulation was extended in [74] and [75], focusing on the error mitigation properties. Both works not just manage to eliminate the blindness against certain type of Pauli errors, but even define distance three codes with which all Pauli errors of weight one can be corrected. Note that the goal of error mitigation and even correction is somewhat orthogonal to locality efforts. For quantum error correction the code distance, the minimal weight of a logical operator, is sought to be increased, whereas it is sought to be minimized for locality. However, the construction of [75], called *Generalized Superfast Encoding* (GSE) turns out to be a versatile scheme that can be used for either purpose. For the sake of completeness within this work, let us quickly sketch the idea of this mapping. In the GSE, the entirety of qubits is sectioned into partitions $n_1, n_2, \dots, n_N$, for each mode, where each partition n_j has $d_j/2$ qubits, with d_j being the number of edges at node (mode) j that the partition represents. In [75], a different encodings for the $\mathcal{A}$- and $\mathcal{B}$-operators are found relying on Majorana-like Pauli strings on every node. The strings of one node, $\{\gamma^k_{n_j}\}_{k\in[d_j]}$, only anticommute with each other, not with the γ-operators of a different node. Therefore, we can think of every node as hosting a local set of Majoranas distinctly encoded on the qubits at the node. An operator $\mathcal{A}_{jk}$ now is defined not just with the direction ε_{jk}, but also which two γ-strings on nodes j and k are associated with the edge. Any γ-string can only participate in one edge operator, and so we say that for an edge jk, we select the Majoranas s and t from nodes j and k. The logical operators are defined as

$$\mathcal{A}_{jk} = \varepsilon_{jk}\, \gamma^s_{n_j} \otimes \gamma^t_{n_k} \tag{3.69}$$

$$\mathcal{B}_k = (-i)^{d_k/2} \prod_{m=1}^{d_k} \gamma^m_{n_k}\,, \tag{3.70}$$

and the stabilizers are defined as before. In contrast to the BKSF, an interaction graph underlying the GSE is only valid if every of its vertices has even degree. That also means that there is an Euler path connecting every vertex using each edge exactly once. The Euler path is important, since the multiplication of $\mathcal{A}$-operators along yields a stabilizer proportional to $\prod_k \mathcal{B}_k$, the parity operator. It is thus the proportionality constant ± 1 that determines the total parity of the system simulated.
To simulate systems that do not comply with the even-degree condition on the interaction graph, the introduction of dummy edges and vertices is advertised in [75]. Even more conditions on the graph connectivity have to be imposed in order to achieve a code distance of three, which would allow for quantum error correction. However, for the task of this chapter, we can also try to cast this mapping into a square lattice of qubits. Considering that the interaction graph is a square lattice, the total number of qubits would be $2\ell_1\ell_2 - 4$, where it is necessary to introduce ears at the boundaries such that every node has two qubits associated with it. The only exceptions are the nodes in the lattice corners, which only have one. Ideally, one would like to define

$$\{\gamma^1_{n_j}, \gamma^2_{n_j}, \gamma^3_{n_j}, \gamma^4_{n_j}\} \;=\; \{X \otimes \mathbb{I},\, Y \otimes \mathbb{I},\, Z \otimes X,\, Z \otimes Y\} \tag{3.71}$$

on every node, where the first two operators are chosen e.g. for the two $\mathcal{A}$-operators in horizontal direction. As a consequence, a Manhattan string of $\mathcal{A}_{(i,\,j)(i+x,\,j+y)}$ would roughly be of weight $x + y$. However, assigning the qubits to a square lattice, such that all $\mathcal{A}_{(i,\,j)(i+x,\,j+y)}$ are continuous, seems impossible. Instead, we can employ a different choice of (3.71) and settle for weight $x + 2y$, but with continuous strings, a planar code on the square lattice with weight-6 stabilizers. This choice is characterized by the definition of

$$\{\gamma^1_{n_j}, \gamma^2_{n_j}, \gamma^3_{n_j}, \gamma^4_{n_j}\} \;=\; \{X \otimes Y,\, Y \otimes Y,\, \mathbb{I} \otimes X,\, Z \otimes Y\} \tag{3.72}$$

on every node j. In conclusion, there is a Superfast mapping with Manhattan distance strings of one preferred direction like in the AQM and VCT. Its qubit requirements are similar to the VCT, and $\mathcal{B}$-strings are encoded also by a singular Z-operator. After its state preparation, classical side computations have to aid the attachment of minus signs to $\mathcal{A}$- and $\mathcal{B}$-operators, such that the targeted parity subspace is encoded. Note that while the representation (3.72) guarantees continuous strings for long-range interactions, (3.71) can be used for error mitigation purposes, as it forms a weight-2 code.

Chapter 4

Quantum error correction in Crossbar architectures

4.1 Background

Spin qubits in silicon quantum dots are a promising platform for quantum computation. Isotopically enriched silicon-28 not only promises long coherence times, but also the compatibility with semiconductor manufacturing techniques. Offering a high qubit density, silicon quantum dots are an important chance for the future of quantum computing. However, controlling a vast amount of qubits is nontrivial. Out of an array of $N \times N$ quantum dots, we would have to be able to select singular qubits for quantum gates and measurements. Those operations can be performed by manipulating electric potentials on or around the corresponding dots, for which gate lines have to connect the corresponding elements with a classical interface. This typically means to steer $O(N^2)$ elements in the bulk of the dot array from its boundary, which only has space to connect $O(N)$ gate lines. The solution for this mismatch is known from classical electronics: a crossbar switch allows to address single elements in a matrix of components by the use of certain row and column lines. For the quantum case, a similar strategy can be adopted, and so only $O(N)$ gate lines are necessary to control the entire grid. The idea is that grid operations like quantum gates and measurements only happen where pulsed lines connect to the same physical elements, such that individual qubit control is achievable. The price to pay for this is a reduced ability to perform operations on different units in the grid in parallel. For classical systems this is not a fundamental problem, but when the computational units are qubits, whose information decays over time, parallelism becomes absolutely essential. This introduces a formidable roadblock for the development of crossbar systems for quantum computing systems. Nevertheless various crossbar architectures for quantum computers have been proposed in the past [18, 76–79]. This chapter is focusing on our proposal for crossbar-controlled spin quantum dots in silicon [18].

Any realistic quantum computing device, including the one we propose in [18], will suffer from noise processes that degrade quantum information. This noise can be combated by quantum error correction [80, 81], where quantum information is encoded redundantly in such a way that errors can be diagnosed and remedied as they happen without disturbing the encoded information. Many quantum error correction codes have been developed over the last two decades and several of them have desirable properties such as high noise tolerance, efficient decoders and reasonable implementation overhead. Of particular note are the planar surface [82] and color codes [83], which can be implemented in quantum computing systems in which only nearest-neighbor two-qubit gates are available.

However these codes, and all other quantum error correction codes, are developed under the assumption that all physical qubits participating in the code can be controlled individually in parallel. While practical large-scale quantum computers most likely pose control limitations, surprisingly little work has been done in this area [72]. Here we investigate the minimal amount of parallel control resources needed for quantum error correction in the proposed architecture [18].

4.2 Results

- We analyze the crossbar architecture we propose in [18]. We give a full description of the layout and control characteristics of the architecture in a manner accessible to non-experts in quantum dots. We develop a language for describing operations in the crossbar system. Of particular interest here are the regular patterns (see e.g. Section 4.4.4) that are implied by the crossbar structure. These configurations provide an abstraction on which we build mappings of quantum error correction codes (see below) This analysis is particular to the system in [18] but we believe many of the considerations to hold for more general crossbar architectures.

- We map the planar surface code and the 6.6.6. (hexagonal) and 4.8.8. (square-octagonal) color codes [83] to the crossbar architecture, taking into account its limited ability to perform parallel quantum operations. The tools we develop for describing the mapping, in particular the configurations described in Section 4.4.4, should be generalizable to other quantum error correction codes and general crossbar architectures.

- Due to experimental limitations the mappings mentioned above might not be attainable in near term devices. Therefore we adapt the above mappings to take into account practical limitations in the architecture [18]. In this version of the mapping the length of an error correction cycle scale with the distance of the mapped code. This means the mapping does not allow for arbitrary logical error rate suppression. Therefore we analyze the behavior of the logical error rate with respect to estimated experimental error parameters and find that the logical error rate can in principle be suppressed to below 10^{-20} (an error rate comparable to the error rate of classical computers [84]), allowing for practical quantum computation to take place.

- Our work raises several interesting theoretical questions regarding the mapping of quantum algorithms to limited control settings, see Section 4.7.

In Section 4.3 we introduce the architecture proposed in [18]. We forgo a deeper discussion of the device physics and only regard its peculiarities as abstract control aspects. We aim to explain the operation of the device in a largely self-contained manner accessible to non experts in quantum dot physics. For that purpose introduce classical helper objects such as the BOARDSTATE matrix which will aid later developments. We discuss one- and two-qubit operations, measurements, and qubit shuttling. In Section 4.4 we focus on difficulties inherent in parallel operation

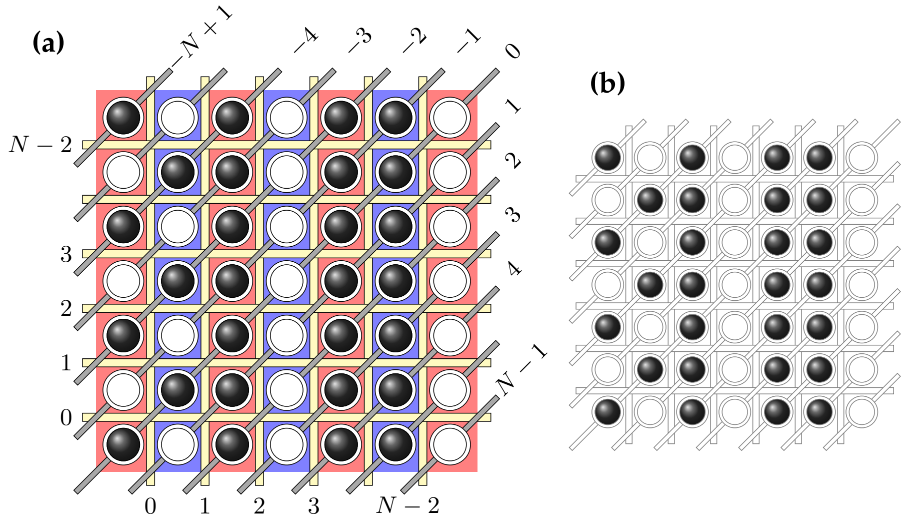


Figure 4.1. (a) A schematic of the Quantum Dot Processor (QDP) that we propose [18], see Section 4.3.1 for details. The white circles correspond to quantum dots, with the black filling denoting the presence of electrons, whose spins are employed as qubits. All dots are found in either red or blue columns, representing areas of different magnetic field. Single qubit gates can only be applied globally on either all qubits in all blue columns or all qubits in all red columns. The vertical, horizontal (both yellow) and diagonal lines (gray) are a feature of this crossbar scheme. The horizontal and vertical gate lines implement barriers that isolate the dots from each other. The diagonal lines simultaneously control the dot potentials of all dots coupled to one line. Quantum operations are effected by pulsing individuals lines. In order to perform two-qubit operations on adjacent dots, one typically needs to lower the barrier that separates them and change the dot potentials by operating the diagonal lines. Note that two-qubit gates applied to adjacent qubits in the same column are inherently different (by nature of the QDP design) from two-qubit gates between two adjacent qubits in the same row. With the control lines, we can also move qubits from dot to dot and measure them. However, since each control line influences $O(N)$ qubits, individual qubit control, as well as parallel operation on many qubits is limited. **(b)** Abstracted version of the QDP scheme representing the classical BOARDSTATE matrix. The BOARDSTATE holds no quantum information, but encodes where qubits are located on the QDP grid.

of the crossbar system. We also introduce several BOARDSTATE configurations which feature prominently in quantum error correction mappings. We describe how these configurations can be reached efficiently by parallel shuttling. In Section 4.5.3 we bring together all previous sections and devise a mapping of the planar surface code to the crossbar architecture. This we continue in Section 4.5.4 for the $6.6.6.$ and $4.8.8.$ color codes. Finally in Section 4.6 we analyze in detail the logical error probability of the surface code mapping as a function of the code distance and estimated error parameters of the crossbar system.

4.3 The quantum dot processor

In this section we will give an overview of the quantum dot processor (QDP) architecture as proposed in [18]. Although this chapter considers the concrete realization of quantum dots in silicon, our main focus is going to lie on its crossbar control structure. Therefore, we will not engage too much with the physics of the host system, but abstract its peculiarities into operational properties as they are relevant for our purposes of controlling this device. The basic organization of the QDP is that for an $N \times N$ grid of qubits interspersed with control lines that effect operations on the qubits. The most notable feature of the QDP (and crossbar architectures in general) is the fact that any classical control signal sent to a control line will be applied simultaneously to all qubits along it. This means that every possible classical instruction applied to the QDP will affect $O(N)$ qubits (these qubits will not necessarily be physically close to each other). This has important consequences for the running of quantum algorithms on the QDP (or any crossbar architecture) that must be taken into account when compiling these algorithms to hardware level instructions. Notably it places strong restrictions on performing quantum operations in parallel on the QDP. To deal with these restrictions it is important to have an understanding of how operations are performed on the QDP. For this reason that we begin our study of the QDP with an examination of its control structure at the hardware level. We describe the physical layout of the system and develop nomenclature for the fundamental control operations. This nomenclature might be called the 'machine code' of the QDP. From these basic instructions we go on to construct all elementary operations that can be applied to qubits. These are quantum operations, such as single qubit gates, nearest-neighbor two-qubit gates and qubit measurements but also a non-quantum operation called coherent shuttling which does not affect the quantum state of the QDP qubits but changes their connectivity graph (i.e. which qubits can be entangled by two-qubit gates). All of these operations are restricted by the nature of the control architecture in a way that gives rise to interesting patterns (Section 4.4.4) and which we will fully examine in Section 4.4.

4.3.1 Layout

A schematic overview of the QDP architecture is given in Figure 4.1, where qubits (which are electrons, denoted by black balls) occupy an array of $N \times N$ quantum dots. The latter are denoted by white dots when empty, since they either are occupied by a qubit or not. We will label the dots by tuples containing row and column indices $(i, j) \in [0 : N - 1]^{\otimes 2}$ beginning from the *bottom left* corner[1]. We assume all qubits to be initialized in the state $|0\rangle$. For future reference we note that $|0\rangle$ corresponds to the spin-up state and $|1\rangle$ to the spin-down state of the electron constituting the qubit.

Typically we will work in a situation where half the dots are occupied by a qubit and half the dots are empty (as seen in Figure 4.1 (a)). Because (as we discuss in Section 4.3.3.1) the qubits can be moved around on the grid, it is important to keep track of which dots contain qubits and which ones do not. This can be done efficiently in classical side-processing. To this end we introduce the BOARDSTATE object. BOARDSTATE consists of a binary $N \times N$ matrix with a 1 as the (i, j)-th entry if the (i, j)-th dot contains an electron and 0 otherwise. The BOARDSTATE does not contain information about the qubit state, only about the electron occupation of the grid. A particular BOARDSTATE is illustrated in the left panel of Figure 4.1.

We now turn to describing the control structures that are characteristic for this architecture. As a first feature, we would like to point out that each dot is either located in a red or a blue region in Figure 4.1 (left panel). The blue (red) columns correspond to regions of high (low) magnetic fields, which plays a role in the addressing of qubits for single qubit gates. We

[1] This is a difference from last chapter's notation, where we started counting from 1, and the components of the index appeared in the opposite order to resemble euclidean coordinates.

will denote the set of qubits in blue columns (identified by their row and column indices) by $\mathcal{B}$ and the set of qubits in red columns by $\mathcal{R}$.

Much finer groups of dots can be addressed by the control lines that run through the grid. The crossbar architecture features control lines that are connected to $O(N)$ dots. At the intersections of these control lines individual dots and qubits can be addressed. This means that using $O(N)$ control lines $O(N^2)$ qubits can be controlled. As seen in Figure 4.1 the rows and columns of the QDP are interspersed with horizontal and vertical lines (yellow), as a means to control the tunnel coupling between adjacent dots. We refer to those lines as barrier gates, or barriers for short. Each line can be controlled individually, but a pulse has an effect on all $O(N)$ dot pairs it separates. Another layer of control lines is used to address the dots itself rather than the spaces in between them. The diagonal gate lines (gray), are used to regulate the dot potential. We label the horizontal and vertical lines by an integer running from 0 to $N-2$ and the diagonal lines with integers running from $-N+1$ to $N-1$ where the $-(N-1)$-th line is the top-left line and increments move towards the bottom right (see Figure 4.1(a)). We count horizontal and vertical lines starting at zero from the lower left corner of the grid (see Figure 4.1). Note that the barriers at the boundary of the grid are never addressed in our model and are thus not labeled. Next we describe how all control lines can be used to effect operations on the qubits occupying the QDP grid.

4.3.2 Control and addressing

As described above, the QDP consists of quantum dots interspersed with barriers and connected by diagonal lines. For our purposes these can be thought of as abstract control knobs that apply certain operations to the qubits. In this section we will describe what type of gates operations are possible on the QDP. We will not concern ourselves with the details of parallel operation until Section 4.4.

There are three fundamental operations on the QDP which we will call the “grid operations”. These operations are “lower vertical barrier” (V), “lower horizontal barrier” (H) and “set diagonal line” (D). The first two operations are essentially binary (on-off) but the last one (D) can be set to a value $t \in [0 : T]$ where T is a device parameter. At the physical

OPCODE	Effect
V$[i]$	Lower vertical barrier at index i
H$[i]$	Lower horizontal barrier at index i
D$[i][t]$	Set diagonal line at index i to value t

Table 4.1. Table of grid operations.

level this corresponds to how many clearly distinct voltages we can set the quantum dot plunger gates [18]. Although the actual pulses on those gates differ by amplitude and duration between the different gates and operations, this notation gives us a clear idea which lines are utilized. This can be done because realistically one will not interleave processes in which pulses have such different shapes. We can label the grid operations by mnemonics (which in a classical analogy we will call OPCODES) as seen in Table 4.1. These OPCODES are indexed by an integer parameter that indicates the label of the control line it applies to.

We indicate parallel operation of a collection of OPCODES by ampersands, e.g. D[1]&H[2]&D[5]. The three grid operations are summarized in Table 4.1. These grid operations can be used to induce some elementary quantum gates and operations on the qubits in the QDP. Below we describe these operations.

4.3.3 Elementary operations

Here we give a short overview of the elementary operations available in the QDP. We will describe basic single qubit gates, two-qubit gates, the ability to move qubits around by coherent shuttling [20] and a measurement process through Pauli Spin Blockade (PSB) [85]. All of these operations are implemented by a combination of the grid operations defined in Table 4.1, and are inherently dependent on the BOARDSTATE .

4.3.3.1 Coherent qubit shuttling

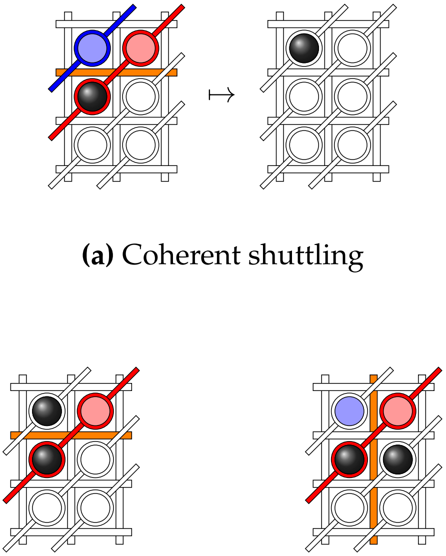
An elementary operation of the QDP is the coherent qubit shuttling [20, 86], of one qubit to an adjacent, empty dot. That means that an electron (qubit) is physically moved to the other dot utilizing at least one diagonal line and the barrier between the two dots. It thereby does not play a role whether the shuttling is in horizontal (from a red to a blue column or the other way around) or vertical direction (inside the same column). However, the shuttling in between columns results in a Z-rotation, that must be compensated by timing operations correctly, see [18] for details. This Z-rotation can also by used as a local single qubit gate, see Section 4.3.3.3. The operation is dependent on the BOARDSTATE by the prerequisite that the dot adjacent to the qubit to must be empty. Collisions of qubits are to be avoided, as those could lead to the formation of different charge states (see however the measurement process in Section 4.3.3.2). We now describe the coherent shuttling as the combination of grid operations.

We lower the vertical (or horizontal) barrier in between the two dots and instigate a 'gradient' of the on-site potentials of the two dots. That is, the diagonal line of the dot containing the qubit must be operated at $t \in [0 : T]$ while the line overhead the empty dot must have the potential $\hat{t} \in [0 : T]$ with $\hat{t} = t - 1$. Note that this implies it might not be operated at all (if it is already at the right level). We will subsequently refer to the combination of a lowered barrier and such a gradient as a "flow". A flow will in general be into one of the four directions on the grid. We define the commands $\text{VS}[i, j, k]$ (vertical shuttling) and $\text{HS}[i, j, k]$ (horizontal shuttling). The command $\text{VS}[i, j, k]$ shuttles a qubit at location (i, j) to $(i + 1, j)$ for $k = 1$ (upward flow) and shuttles a qubit at location $(i + 1, j)$ to (i, j) for $k = (-1)$ (downward flow). Similarly, the command $\text{HS}[i, j, k]$ shuttles a qubit at location (i, j) to $(i, j+1)$ for $k = 1$ (rightward flow) and shuttles a qubit at location $(i, j + 1)$ to (i, j) for $k = (-1)$ (leftward flow). See Table 4.2 for a summary of these OPCODES.

Using only these control lines, we can individually select a single qubit to be shuttled. However, when attempting to shuttle in a parallel manner, we have to be carefully take into account the effect that the activation of several of those lines has on other locations. We will deal with this in more detail in Section 4.4.1.

4.3.3.2 Measurement and readout

The QDP allows for local single qubit measurements in the computational basis $|0\rangle, |1\rangle$. We can measure a qubit by using essentially the same lines as if we were to shuttle it to a horizontally adjacent dot that is already occupied by a qubit in a fixed state of reference: that qubit will



(a) Coherent shuttling

(b) $\sqrt{\text{SWAP}}$

(c) CPhase* or measurement

Figure 4.2. Schematic representation of the use of control lines for the native operations in the QDP. Qubits are represented by black balls on the grid. Red or blue colored dots are empty, but their dot potentials change due to an operation of the diagonal line they are coupled to. Empty dots, unaffected by grid operations, are white. **(a)** Vertical shuttling of a qubit (to the top left dot) requires to lower the orange barrier. One can than either raise the dot potentials on the red diagonal line, or lower the potential on the blue dot by pulsing the blue diagonal. **(b)** Schematic representation of the control lines used for performing two-qubit $\sqrt{\text{SWAP}}$ gate between the two qubits on that grid. The orange barrier is lowered and the red diagonal line is utilized to detune dot potentials. **(c)** Grid operations necessary to perform a measurement or a two-qubit effective CPhase* gate between the two qubits. The orange barrier between the two qubits is lowered, and the dot potentials along the red diagonal line is raised by pulsing the latter. Note that the empty, red colored dot is also effected by that action, and its barrier to the adjacent dot is lowered. If the two dots in the upper row were not empty, side effects would occur. See Section 4.3.3.4 for more information on the nature of the two-qubit gates. Note also that the readout procedure of the measurement requires us to have the upper dot (light blue) empty, if the barrier gate between them is used for readout.

therefore be referred to as reference qubit.[2] When the first qubit is in contact with the reference, their total spin wavefunction collapses into either a singlet or a triplet state. Due to the Pauli principle, those two spin wave functions produce an antisymmetric spatial wave function resulting in a different distribution of charge over the two dots. This charge distribution can be detected in the readout. This process is called Pauli Spin Blockade (PSB) measurement [18, 85]. However, the QDP's ability to perform this type of qubit measurements is limited by three factors.

Firstly, the measurement requires a reference qubit horizontally adjacent to the qubit to be measured. Not only must the reference be in a known computational basis state, but the choice of state depends on the magnetic field, i.e. whether the dots are in red or blue columns in Figure 4.1(a). A reference qubit in the set $\mathcal{B}$ must be in the state $|0\rangle$, whereas one found in $\mathcal{R}$ must be in $|1\rangle$. The qubit that is to be measured, has to be in the respective other column, vertical measurements are not allowed. This effectively means that when a qubit pair is in the wrong configuration we must first shuttle both qubits one step to the left (or to the right). Note that this takes two additional shuttling operations, which means it is important to keep track at all times where the two qubits are on the BOARDSTATE, or else incur a shuttling overhead (which might become significant when dealing with large systems and many simultaneous measurements). We will deal with the problem of qubit-pair placement in more detail in Section 4.4.3.

Secondly, assuming that the qubit pair is in the right configuration to perform the PSB process, one still needs to perform a shuttling-like operation to actually perform the measurement. On the technical level, the operation is different from coherent shuttling, but the use of the lines is similar with the difference that after the readout, the shuttling-like operation is undone by the use of the same lines as before - which are not necessarily the lines one would use to reverse a coherent shuttling operation. However, scheduling measurement events on the QDP is at least as hard as the scheduling of shuttle operations discussed above. Depending on the state the qubit is in, it will now assume one of two possible states that can be distinguished by their charge distribution.

[2] Not to be confused with the measurement qubit in the surface code. The role of the latter is assumed by the first qubit, that is supposed to be measured.

OPCODE	Control OPCODES	Effect
$\mathrm{HS}[i,j,k]$	$\mathrm{V}[i]$ & $\mathrm{D}[i-j][t-1/2-k/2]$ & $\mathrm{D}[i-j+1]$[t-1/2+k/2]	$(k=1)$: Shuttle from (i,j) to $(i,j+1)$ $(k=-1)$: Shuttle from $(i,j+1)$ to (i,j)
$\mathrm{VS}[i,j,k]$	$\mathrm{H}[j]$ & $\mathrm{D}[i-j][t-1/2-k/2]$ & $\mathrm{D}[i-j-1][t-1/2+k/2]$	$(k=1)$: Shuttle from (i,j) to $(i+1,j)$ $(k=-1)$: Shuttle from $(i+1,j)$ to (i,j)
$\mathrm{M}[i,j,k]$	$\mathrm{HS}[i,j+1/2+k/2,-k]$	Measurement of qubit (i,j) using the qubit at $(i,\ j+k)$

Table 4.2. OPCODES for horizontal and vertical shuttling and measurement together with the control OPCODES required to implement these operations on the QDP.

Thirdly, the readout process requires to have a barrier line that borders to the qubit pair, with an empty dot is across the spot of the qubit to be measured. This is a consequence of the readout procedure [18].

In Table 4.2 we introduce the measurement OPCODE $\mathrm{M}[i,j,k]$ with $k \in \{-1,1\}$ to denote a measurement of a qubit at location (i,j) with a reference located to the left $(k=-1)$ or to the right $(k=1)$.

4.3.3.3 Single-qubit rotations

There are two ways in which single qubit rotations can be performed on the QDP, both with drawbacks and advantages. The first method, which we call the semi-global qubit rotation, relies on electron-spin-resonance [87]. Its implementation in the QDP allows for any rotation in the single qubit special unitary group $SU(2)$ [88] to be performed but we do not have parallel control of individual qubits. The control architecture of the QDP is such that we can merely apply the same single qubit unitary rotation on all qubits in either $\mathcal{R}$ or $\mathcal{B}$ (even or odd numbered columns). Concretely we can perform in parallel the single qubit unitaries

$$U_{\mathcal{R}} = \bigotimes_{(i,j)\in\mathcal{R}} U_{(i,j)}\,, \qquad U_{\mathcal{B}} = \bigotimes_{(i,j)\in\mathcal{B}} U_{(i,j)} \tag{4.1}$$

where $U_{(i,j)}$ means applying the same unitary U to the state carried by the qubit at location (i,j). In general the only way to apply an arbitrary single qubit unitary on a single qubit in $\mathcal{B}$ (or $\mathcal{R}$) is by applying the unitary to all qubits in $\mathcal{B}$ ($\mathcal{R}$), moving the desired qubit into an adjacent column, i.e. from $\mathcal{B}$ to $\mathcal{R}$ ($\mathcal{R}$ to $\mathcal{B}$) and then applying the inverse of the target unitary to $\mathcal{R}$ ($\mathcal{B}$). This restores all qubits except for the target qubit to their original states and leaves the target qubit with the required unitary applied. The target qubit can then be shuttled to its original location. A graphical depiction of the BOARDSTATE associated with this maneuver can be found in Figure 4.3. This means applying a single unitary to a single qubit takes a constant amount of grid operations regardless of grid size.

The second method does allow for individual Z-rotations on single qubits: $\exp(i\phi Z) = \cos\phi \cdot \mathbb{I} + i \sin\phi \cdot Z$. This operation can be performed on a given qubit at (i, j) by shuttling it to an empty dot at $(i, j \pm 1)$ (and perhaps back). When the qubit leaves the column it was originally defined on ($\mathcal{B}$ to $\mathcal{R}$ or vice versa) it will effectively start precessing about its Z-axis [18]. This effect is always present but it can be mitigated by timing subsequent operations such that a full rotation happens between every operation (effectively performing the identity transformation, see Section 4.3.3.1). By changing the timing between subsequent operations any rotation angle ϕ can be effected. This technique will often be used to perform the Z-gate ($\phi = \pi/2$) and the $S = \sqrt{Z}$ phase gate ($\phi = \pi/4$) in error correction sequences.

4.3.3.4 Two-qubit gates

As the last elementary tool, we have the ability to apply entangling two-qubit gates on adjacent qubits. The QDP can perform two different types of two-qubit gates. Inside one column, so between qubits at locations (i, j) and $(i \pm 1, j)$, a square-root of SWAP ($\sqrt{\text{SWAP}}$) can be realized [89]. This can be done by lowering the horizontal barrier between the two qubits and toggling the voltage on the diagonal lines overhead the two qubits. This situation is illustrated in Figure 4.2 (c). The $\sqrt{\text{SWAP}}$ gate is defined as

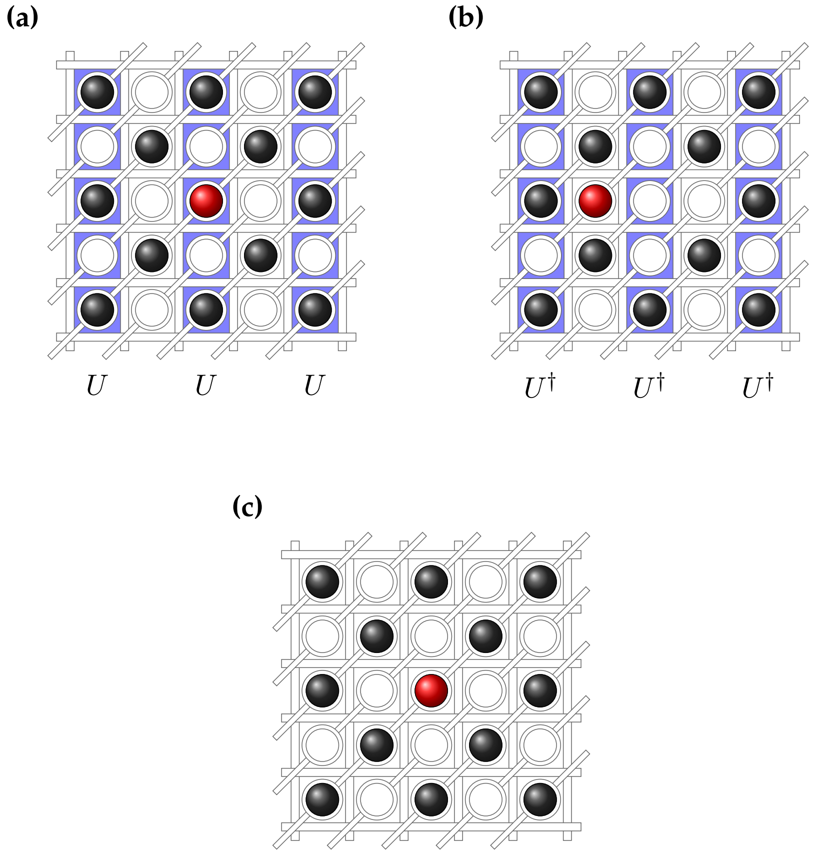


Figure 4.3. BOARDSTATE schematic for applying the unitary U to a single qubit (red). Time flows from (a) to (c) in this schematic. This process illustrates both, the possibility to retain single qubit control by using coherent shuttling, and the overhead that comes with it. In **(a)** we firstly apply the unitary U (blue bars) to all qubits in $\mathcal{R}$ ($\mathcal{B}$). We then move the qubit to the adjacent column. Note that this takes two operations because we do not want any other qubits transitioning with it. In **(b)**, we apply the inverse unitary $U^\dagger$ to all qubits in $\mathcal{R}$ ($\mathcal{B}$). In the last step we move the red qubit back, such that it is in its original position in **(c)**.

$$\sqrt{\textsc{Swap}} = \begin{pmatrix} 1 & & & \\ & (1+i)/2 & (1-i)/2 & \\ & (1-i)/2 & (1+i)/2 & \\ & & & 1 \end{pmatrix}, \tag{4.2}$$

in the computational basis, and has the name-lending property $\sqrt{\textsc{Swap}} \cdot \sqrt{\textsc{Swap}} = \textsc{Swap}$. Alternatively, between horizontally adjacent qubits, e.g. between $(i, j) \in \mathcal{R}$ and $(i, j \pm 1) \in \mathcal{B}$ the native two-qubit gate is an effective CPHASE gate which acts on the computational basis as

$$\text{CPHASE}^{\star} = \begin{pmatrix} 1 & & & \\ & e^{i\phi_1} & & \\ & & e^{i\phi_2} & \\ & & & 1 \end{pmatrix}, \tag{4.3}$$

where the two angles obey $(\phi_1 + \phi_2 \mod 2\pi) = \pi$ (demonstrated in [90–92]). This gate can be performed between horizontally adjacent qubits by lowering the vertical barrier between them and toggling the overhead diagonal lines. This is illustrated in Figure 4.2(a). The CPHASE* can be corrected to a CPHASE by the readily available methods of performing individual Z-rotations. In practice, however, we expect the $\sqrt{\textsc{Swap}}$ gate to have significantly higher fidelity than the CPHASE* gate [18], so in any application (e.g. error correction) the $\sqrt{\textsc{Swap}}$ gate is the preferred native two-qubit gate on the QDP. In Table 4.3 we define OPCODES for the horizontal interaction (CPHASE*) and the vertical interaction ($\sqrt{\textsc{Swap}}$).

4.3.3.5 CNOT subroutine

Many quantum algorithms are conceived using the CNOT gate as the main two-qubit gate. However the QDP does not support the CNOT gate natively. It is easy to construct the CNOT gate from the CPHASE* gate by dressing the CPHASE gate with single qubit Hadamard rotations as seen in Figure 4.4(center). It is slightly more complicated to construct a CNOT gate using the $\sqrt{\textsc{Swap}}$ but it can be done by performing two $\sqrt{\textsc{Swap}}$ gates interspersed single qubit rotations [91–93] as seen in Figure 4.4(right). If the control qubit is moved from an adjacent column on the QDP (as it is in most cases we will deal with) the Z- and S-gates can be performed by the Z-rotation-by-waiting technique described in

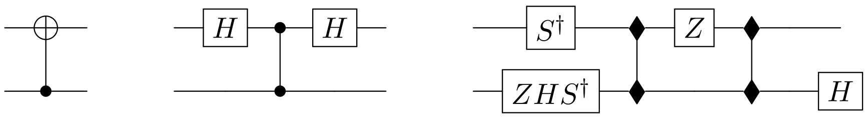


Figure 4.4. Construction of the CNOT gate out of the native CPHASE* and $\sqrt{\text{SWAP}}$ gates. Note that one requires two $\sqrt{\text{SWAP}}$ gates to construct a CNOT gate [93]. When performing arbitrary algorithms it would be preferable to forgo this substitution and instead compile the algorithm directly into a gate-set containing the $\sqrt{\text{SWAP}}$ gate.

OPCODE	Effect	Parameter
HI$[i,j]$	Perform CPHASE* gate between dots (i,j) and $(i,j+1)$	$(i,j) \in [0:N-2]^{\otimes 2}$
VI$[i,j]$	Perform $\sqrt{\text{SWAP}}$ gate between dots (i,j) and $(i+1,j)$	$(i,j) \in [0:N-2]^{\otimes 2}$
HC$[i,j]$	Perform CNOT (using CPHASE*) between (i,j) and $(i,j+1)$	$(i,j) \in [0:N-2]^{\otimes 2}$
VC$[i,j]$	Perform CNOT (using $\sqrt{\text{SWAP}}$) between (i,j) and $(i+1,j)$	$(i,j) \in [0:N-2]^{\otimes 2}$

Table 4.3. OPCODES for horizontal and vertical two-qubit operations on the QDP, respectively the CPHASE* and $\sqrt{\text{SWAP}}$ gates. We also include OPCODES for the performing of CNOT gates composed of $\sqrt{\text{SWAP}}$ or CPHASE* gates.

the last section. For completeness we also define an OPCODE for the CNOT operation in Table 4.3.

4.4 Parallel operation of a crossbar architecture

In this section we focus on performing operations in parallel on the QDP (or more general crossbar architectures). Because of the limitations imposed by the shared control lines of the crossbar architecture, achieving as much parallelism as possible is a nontrivial task. We will discuss parallel shuttle operations, parallel two qubit gates, parallel single qubit gates and parallel measurement. As part of the focus on parallel shuttling we also include some special cases relevant to quantum error correction where full parallelism is possible.

Before we start our investigation however, we would like to put three issues into focus that are likely to be encountered when attempting parallel operations. Firstly, it must be understood that an operation on one location on a crossbar system can cause unwanted side effects in other locations (that might be far away). As indicated in Section 4.3 many elementary operations on the grid in particular take place at the crossing points of control lines. This means that any parallel use of these grid operations must take into account "spurious crossings" which may have such unintended side effects. Let us illustrate such a spurious crossing with an example. Imagine we want to perform the vertical shuttling operations $\mathrm{VS}[i, j-1, 1]$ and $\mathrm{VS}[i+2, j-1, 1]$ in parallel (see Figure 4.5 for illustration). We can do this by lowering the horizontal barriers at rows i and $i+2$ (orange in illustration) and elevating the on-site potentials on the diagonal lines $i-j+1$ and $i+2-j+1$ (red in illustration). This will open upwards flows at locations $(i, j-1)$ and $(i+2, j-1)$. However it will also open an upward flow at the location $(i+2, j+1)$. This means, if a qubit is present at that location an unintended shuttling event will happen. To avoid this outcome we must either perform the operations $\mathrm{VS}[i, j-1, 1]$ and $\mathrm{VS}[i+2, j-1, 1]$ in sequence (taking two time-steps) or perform an operation $\mathrm{VS}[i+2, j+1, -1]$ to fix the mistake we made, again taking two time-steps. This is a general problem when considering parallel operations on the QDP.

Secondly, we would like to point out that in realistic setups, we expect a trade-off between parallelism (manifested in algorithmic depth) and operation fidelity (in particular this will be the case in the QDP system). In order to understand this, we have to be aware that most operations consist of applying the correct pulses for the right amount of time. Due to g-factor variations, these durations can slightly vary from dot to dot. In order to perform the perfect gate, for instance, we must be able to terminate one interaction in a parallel operation. This usually entails being able to eliminate a single crossing by resetting one control line prematurely, if the dot at the crossing has a higher g-factor. If this is not possible (maybe because it would cause side effects) a loss in operation fidelity is a consequence of the resulting improperly timed operation. The most robust case is thus to schedule operations line-by-line. By this we mean that we attempt to perform $O(N)$ grid operations in a single time step while using every horizontal, diagonal or vertical line only once per individual

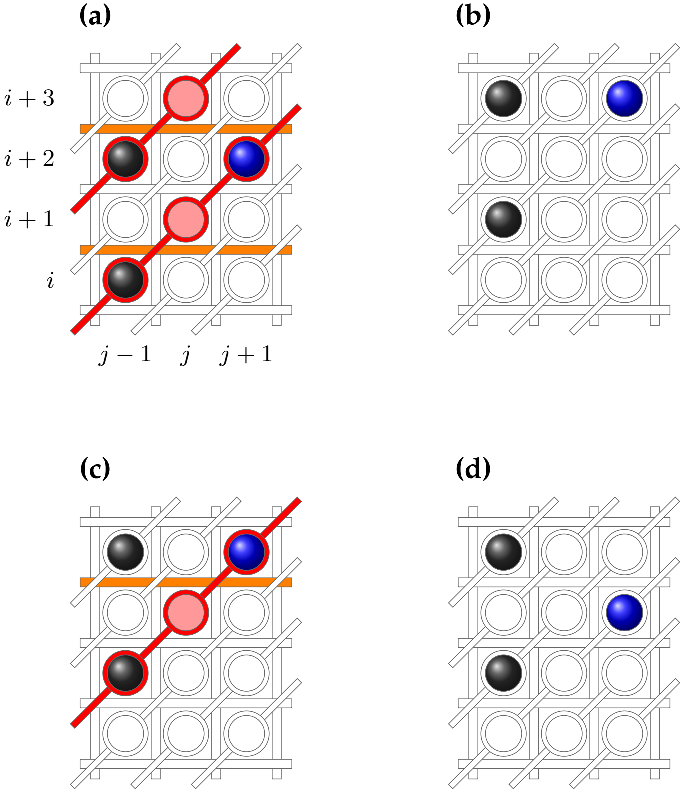


Figure 4.5. Spurious shuttle operations. Here we illustrate an example of unintended side effects that occur due to the limited control. We again denote qubits by colored balls, and color barriers and lines that are operated. Empty dots with changed potentials are colored as well, whereas white dots are unaffected. **(a)** The black qubits are to be shuttled from $(i, j-1)$ to $(i+1, j-1)$ and from $(i+2, j)$ to $(i+3, j)$ respectively without moving the blue qubit. For that purpose, the (orange) barriers between the two dot pairs are lowered, as well as the (red) diagonal lines through $(i, j-1)$ and $(i+2, j)$ are pulsed, such that the dot potentials at those sites are raised. **(b)** The qubit on $(i+3, j+1)$ has unintentionally moved to $(i+2, j+1)$. **(c)** To remedy this situation, we lower the barrier labeled $i+2$ again (orange), and also raise the potential at dot $(i+3, j+1)$ and with it at all other dots that are connected by the pulsed diagonal line (red). In **(d)**, the desired situation is achieved.

grid operation. If we, for instance, schedule several vertical shuttle operations, we may choose to start by lowering the horizontal barrier first and then detune the dot potentials of all qubits adjacent to that barrier, by pulsing the corresponding diagonal lines. To account for the variations, we reset the diagonal lines at slightly different times. Line-by-line operations work with either line types for every two-dot operation (measurement, shuttling and two-qubit gates). Note however that for shuttling operations individual control over one line is sufficient, whereas for measurement and two-qubit gates we would ideally like to be able to control two lines per qubit pair individually, where one line should be the barrier separating the two paired qubits. Results presented in the following take these constraints into account for quantum error correction. The parallel operation nonetheless remains one of the greatest challenges of the crossbar scheme. In this section, we will assume all operations to be perfect (even when performed in parallel) but in Section 4.6 we perform a more detailed analysis of the behavior of the QDP when operational errors are taken into account.

Thirdly, it is important to have access to classical side computations to aid the scheduling of parallel operations without spurious crossings. However, no classical assistance is required for purposes of quantum error correction, such that a discussion of the concrete algorithms is omitted. The interested reader may find an in-depth discussion on the classical side computations within the original work [94] or the crossbar chapter of [95]. As we define parallel versions of the elementary operations in the next step, we would like the reader to bear in mind that these OPCODES work with the classical input, which in our case is however trivial. We begin with discussing parallel shuttle operations.

4.4.1 Parallel shuttle operations

We define parallel versions of the shuttling OPCODES HS$[i, j, k]$ and VS$[i, j, k]$ in the following table.

OPCODE	Effect
HS[L]	Perform HS$[i, j, k]$ for all $(i, j, k) \in$ L
VS[L]	Perform HS$[i, j, k]$ for all $(i, j, k) \in$ L

This OPCODE takes in a set (denoted as L) of tuples (i, j, k) which denote 'locations at which shuttling happens' (i, j) and 'shuttling direction' (k). From these codes it is not immediately clear how many of the shuttling operations can be performed in a single grid operation, i.e. setting the diagonal lines to some configuration and lowering several horizontal or vertical barrier. If multiple grid operations are needed (such as in the example Figure 4.5) we would like this sequence of grid operations to be as short as possible. However, given some initial BOARDSTATE and a parallel shuttling command HS[L] it is not clear what the sequence of parallel shuttling operations actualizing this command is. At the same time, parallelization might not be the ultimate goal, and so other schedules might be implicit in the given OPCODES.

4.4.1.1 Selective parallel single-qubit rotations

In this section we will discuss a particular example that illustrates the use of abstracting away the complexity of parallel shuttling. Imagine a QDP grid initialized in the so called *idle* configuration. This configuration can be seen in Figure 4.6. We will focus on the qubit in the odd columns (i.e. the set $\mathcal{B}$). Imagine a subset S of these qubits to be in the state $|1\rangle$ and the remainder of these qubits to be in the state $|0\rangle$. The qubits on in the set $\mathcal{R}$ can be in some arbitrary (and possibly entangled) multi-qubit state $|\Psi\rangle$. We would like to change the states of the qubits in the set S to $|0\rangle$ without changing the state of any other qubit. Due to the limited single qubit gates (see Section 4.3.3.3) available in the QDP this is a nontrivial problem for some arbitrary set S. However using the power of parallel shuttling we can perform this task as follows. Begin by defining the set of coordinates $\hat{S}$, which hold all qubits in the complement of S in $\mathcal{R}$. Now we begin by performing the parallel shuttling operation

$$\mathrm{HS[L]}, \quad \mathrm{L} = \{(i, j, 1) \ \ || \ \ (i, j) \in \hat{S}\}. \tag{4.4}$$

This operation in effect moves all qubits in $\hat{S}$ out of $\mathcal{R}$ (and into $\mathcal{B}$, note that the dots the qubits are being shuttled in are always empty by the definition of the idle configuration). Now we can use a semi-global single qubit rotation (as discussed in Section 4.3.3.3) to perform X-rotations on all qubits in $\mathcal{R}$, which is at this point all qubits in the set S. These flips change the states of the qubits in S from $|1\rangle$ to $|0\rangle$ without changing the state of any other qubit. Following this we can restore the BOARDSTATE to its original configuration by applying the parallel shuttling command

$$\mathrm{HS[L]}, \quad \mathrm{L} = \{(i, j, -1) \ \ || \ \ (i, j) \in \hat{S}\}. \tag{4.5}$$

OPCODE	Effect
HI[L]	Perform VI$[i,j]$ for $(i,j) \in$ L
VI[L]	Perform HI$[i,j]$ for $(i,j) \in$ L

OPCODE	Effect
VC[L]	Perform VC$[i,j]$ for every (i,j) in L

Now we have applied the required operation. Note that at no point we had to reason about the structure of the set S itself. This complexity was taken care of by the classical subroutines embedded in HS[L]. Next we discuss performing parallel two-qubit gates.

4.4.2 Parallel two-qubit gates

Similar to parallel shuttling it is in general rather involved to perform parallel two-qubit operations in the QDP. We can again define parallel versions of the OPCODES for two-qubit operations and then analyze how to perform them as parallel as possible (again having access to classical side computation).

However, as mentioned before, the parallel operation of two-qubit gates in the QDP will mean taking a hit in operation fidelity vis-à-vis the more controllable line-by-line operation [18]. Since this operation fidelity is typically a much larger error source than the waiting-time-induced decoherence stemming from line-by line operation we will for the remainder of this chapter assume line-by-line operation of the two-qubit gates. This will have an impact when performing quantum error correction on the QDP which we will discuss in more detail in Section 4.6.

For the sake of completeness we also define a parallel version of the CNOT OPCODE. The same considerations of parallel operation hold for the parallel use of CNOT gates as they hold for the CPHASE* and $\sqrt{\text{SWAP}}$ gates. We continue the discussion of parallelism in the QDP by analyzing parallel measurements.

4.4.3 Parallel Measurements

Performing measurements on an arbitrary subset of qubits on the QDP is in general quite involved. Every qubit to be measured requires a ref-

OPCODE	Effect
$\mathrm{M}[\mathrm{L}]$	Perform $\mathrm{M}[i,j,k]$ for every (i,j,k) in L

erence in a known computational basis state, and an empty dot must be adjacent as a reference for the readout process. The qubits must then be shuttled such that the pairs are horizontally adjacent and located in such a way such that they are in the right columns for the PSB process to take place (revisit Section 4.3.3.2 for more information). On top of the required shuttling the PSB process itself (from a control perspective similar to shuttling) must be performed in a way that depends on the BOARDSTATE and the configuration of the reference qubits. In general this PSB process will be performed line-by-line (for the fidelity reasons mentioned in the beginning of the section) and hence requires a sequence of depth $O(N)$ parallel grid operations (plus the amount of shuttling operations needed to attain the right measurement configuration in the first place). Due to this complexity we will not analyze parallel measurement in detail but rather focus on a particular case relevant to the mapping of the surface code. But first we define a parallel measurement OPCODE $\mathrm{M}[\mathrm{L}]$ which takes in a list of tuples (i,j,k) denoting locations of qubits to be measured (i,j) and whether the reference qubit is to its left $(k=-1)$ or to its right $(k=1)$.

4.4.3.1 A specific parallel measurement example

Let us consider a specific example of a parallel measurement procedure that will be used in our discussion of error correction. We begin by imagining the BOARDSTATE to be in the *idle* configuration (Figure 4.6 top left). We next perform the shuttle operations needed to change the BOARDSTATE to the *measurement* configuration. This configuration (and how to reach it by shuttling operations from the idle configuration) will be discussed Section 4.4.4 and can be seen in Section 4.6 (c). Next take the qubits to be measured in the parallel measurement operation to be the red qubits in Figure 4.6. The qubits directly to the right or to the left will serve as a reference (blue in Figure 4.6). We will assume that the reference qubits are in the $|0\rangle$ state. If some of them were in the $|1\rangle$ state instead we would perform the procedure given in Section 4.3.3.3 to rotate them to $|0\rangle$ without changing the state of the other qubits on the grid. With that all the reference qubits are in the set $\mathcal{B}$ and qubits to be read out in the set $\mathcal{R}$, we can perform the PSB process by sending the latter into the dots occupied by the former. Using the shorthand $a \overset{b}{=} c$ to denote $a \bmod b = c$, we employ the commands

$$\text{VS[L]}, \qquad \text{L} = \{(i,j,1) \parallel i \overset{2}{=} 0,\ j \overset{2}{=} 1,\ i+j \overset{4}{=} 1\} \tag{4.6}$$

to bring the qubits to be measured (red) horizontally adjacent to the reference qubits (blue) and then

$$\text{M[L]}, \qquad \text{L} = \{(i,j,1) \parallel i \overset{4}{=} 1,\ j \overset{4}{=} 1\} \tag{4.7}$$

and

$$\text{M[L]}, \qquad \text{L} = \{(i,j,-1) \parallel i \overset{4}{=} 3,\ j \overset{4}{=} 3\}. \tag{4.8}$$

All of these operations can be performed in a single time-step, although the line-by-line manner is preferred by reasons laid out earlier. In particular we would like to perform these operations one row at a time since this gives us the ability to control both diagonal and vertical lines individually for each measurement. However, if we first were to align all pairs, a line-by-line measurement is not possible. For instance when performing measurements on the qubits at locations $(1,1)$ and $(1,5)$ we must measurement is also invoked on the pair at location $(5,5)$. To avoid this situation we will align only the qubits in the bottom row, perform the PSB process and readout on that row only and then undo the shuttlings. This we repeat going up in rows until we reach the end of the grid. More formally we perform the following sequence of operations:

```
For $i \in [0 : N-2]$

    If $i \overset{4}{=} 1$

        VS[L],    $\text{L} = \{(i-1, j, -1) \parallel j \overset{4}{=} 1\}$

        M[L],     $\text{L} = \{(i, j, 1) \parallel j \overset{4}{=} 1\}$

        VS[L],    $\text{L} = \{(i-1, j, 1) \parallel j \overset{4}{=} 1\}$
```

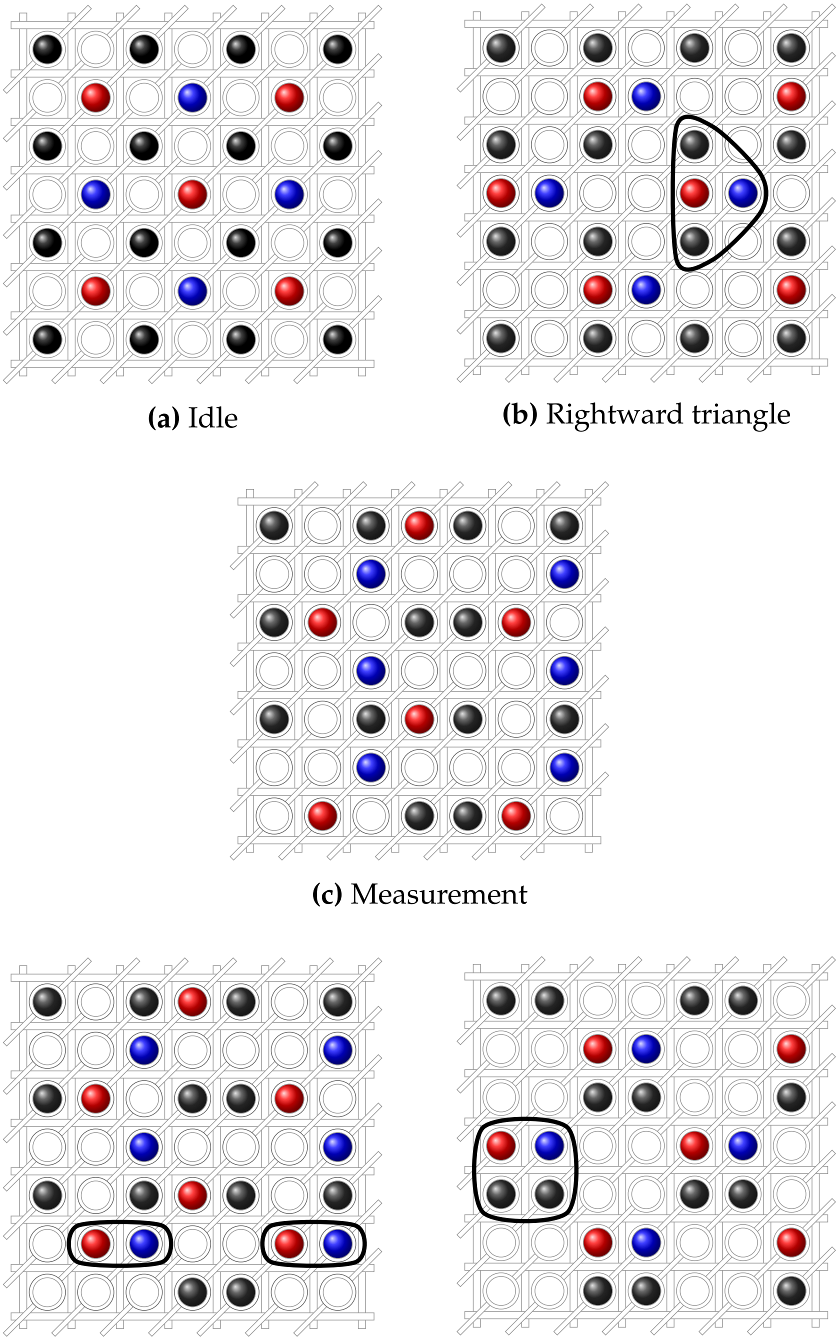
If $i \overset{4}{=} 3$

$$\mathrm{VS[L]}, \qquad \mathrm{L} = \{(i-1, j, \, -1) \parallel j \overset{4}{=} 3\}$$

$$\mathrm{M[L]}, \qquad \mathrm{L} = \{(i, j, \, -1) \parallel j \overset{4}{=} 3\}$$

$$\mathrm{VS[L]}, \qquad \mathrm{L} = \{(i-1, j, 1) \parallel j \overset{4}{=} 3\}\,.$$

We will use this particular procedure when performing the readout step in a surface code error correction cycle in Section 4.5.3. This concludes our discussion of parallel operations on the QDP. We now move on to highlight some BOARDSTATE configurations that will feature prominently in the surface and color code mappings.



(a) Idle

(b) Rightward triangle

(c) Measurement

(d) PSB and readout

(e) Right square

Figure 4.6. Useful BOARDSTATE configurations. We denote memory qubits with dark color, X-measurement qubits by red and Z-measurement qubits by blue. Those will collect the parity of the data qubits in one error correction cycle, and one is the others reference at the PSB measurement. **(a)** The idle configuration is a starting point of all algorithms. All qubits are spread out and well separated. **(b)** The triangle configurations (here we have a rightward triangle, see the frame in the figure) is assumed when the proximity of measurement qubits to data qubits is required. This is the case for the parity measurements in error correction cycles. **(c)** The measurement configuration is formed to bring X- and Z-measurement qubits close to each other, such that a row can be selected in which the measurement is performed. **(d)** Certain measurement qubits are brought to adjacent dots in order to perform the PSB-based measurement and readout in a line-by-line fashion (encircled qubits). Since the rest of the grid is in the measurement configuration, individual control over the barrier lines and one potential is guaranteed without spurious measurements. **(e)** The (right) square configuration is a mid-way point between the idle and (right) triangle configuration. Going through the square configuration keeps the shuttling algorithm manageable, as not more that 2 different heights of the dot potentials are employed. One of the characteristic squares is framed in the figure.

4.4.4 Some useful grid configurations

There are several configurations of the BOARDSTATE that show up frequently enough (for instance in the error correction codes in Section 4.5.3) to merit some special attention. In this section, we list these specific configurations and show how to construct them. Note that this is done using Figure 4.6, in which the red (blue) qubits will later serve as measurement qubits for the $Z-$type ($X-$type) stabilizer tiles of the surface code, while the dark qubits are part of the memory.

4.4.4.1 Idle configuration

The idle configuration is the configuration in which the QDP is initialized. As shown in Figure 4.6, its BOARDSTATE matrix describes a checkerboard pattern. In this configuration no two-qubit gates can be applied between any qubit pair but since it minimizes unwanted crosstalk between qubits [18], it is good practice to bring the system back to this configuration when not performing any operations. For this reason we consider the idle configuration to be the starting point for the construction of all other configurations.

4.4.4.2 Square configuration

As seen in Figure 4.6(e), the square configurations consist of alternating filled and unfilled 2×2 blocks of dots. The so-called right square configuration can be reached from the idle configuration by a shuttling operation HS[L] with the set L being

$$\begin{aligned} \mathrm{L} \ &= \{(i,j,1) \parallel i \overset{2}{=} 1,\ j \overset{2}{=} 1,\ i+j \overset{4}{=} 2\} \\ &\cup\ \{(i,j,-1) \parallel i \overset{2}{=} 0,\ j \overset{2}{=} 1,\ i+j \overset{4}{=} 3\}. \end{aligned} \tag{4.9}$$

Note that this operation only takes a single time-step, and the square configuration is shown in Figure 4.6(e). The right square configuration is characterized by Z-measurement qubits being in the left corner of every square. Another flavor of this configuration is the left square configuration, where the Z-measurement qubits are in the upper right corner, and the X-measurement qubits in the left in the left. The left square configuration can be reached from the idle configuration by a shuttling operation HS[L] with the set L being

$$\begin{aligned}\mathrm{L}\ &= \{(i,j,1) \parallel i \overset{2}{=} 0,\ j \overset{2}{=} 0,\ i+j \overset{4}{=} 2\}\\ &\cup\ \{(i,j,-1) \parallel i \overset{2}{=} 1,\ j \overset{2}{=} 0,\ i+j \overset{4}{=} 1\}.\end{aligned} \tag{4.10}$$

These configurations are used as an intermediate step for us to reach the triangle configurations.

4.4.4.3 Measurement Configuration

The measurement configuration can be reached from the idle configuration in three time-steps by the following sequence of parallel shuttling operations.

$$\begin{aligned}&\mathrm{HS[A]}, && \mathrm{A} = \{(i,j,-1),\ (i-1,j-1,1) \mid i \overset{4}{=} 1,\ j \overset{4}{=} 2\},\\ &\mathrm{HS[B]}, && \mathrm{B} = \{(i-1,j-1,1) \parallel i \overset{4}{=} 3,\ j \overset{4}{=} 1\},\\ &\mathrm{VS[C]}, && \mathrm{C} = \{(i,j,-1) \parallel i \overset{2}{=} 0,\ j \overset{2}{=} 1,\ i+j \overset{4}{=} 1\}.\end{aligned} \tag{4.11}$$

This configuration can be seen in Figure 4.6(d) and it is an intermediate state in the measurement process in which the blue qubits are read out against the red ones. How this measurement protocol works in detail is described in Section 4.4.3.

4.4.4.4 Triangle configurations

In order to collect the parity of memory qubits in the error correction cycles, we need to align the measurement qubits with them, where it hinges on the two-qubit gates whether the alignment is horizontal or vertical. This is reflected in the use of triangle configurations. There are two triangle configurations that can be reached in a single parallel shuttling step from the right square configuration. The first one, seen in Figure 4.6(b), is called the rightward triangle configuration. It can be reached from the square configuration by the grid operation HS[L] with the set L being

$$\mathrm{L} = \{(i,j,-1) \parallel i \overset{2}{=} 1,\ j \overset{2}{=} 1,\ i+j \overset{4}{=} 3\}, \tag{4.12}$$

which does as much as to shuttle the right memory qubit of every square (framed squares in Figure 4.6(e)) to the empty dot on its right. In this configuration, we are able to perform high-fidelity two-qubit gates between measurement and memory qubits in every triangle. In order to reach the neighboring pair of memory qubits, we start from the left square configuration horizontally shuttling the left memory qubit out of every square. Operationally, we would perform $\mathrm{HS[L]}$ with

$$\mathrm{L} = \{(i, j, 1) \;\|\; i \overset{2}{=} 0,\; j \overset{2}{=} 0,\; i + j \overset{4}{=} 2\}. \tag{4.13}$$

Note again that these parallel shuttling operations can be performed in a single time step. From these configurations the idle configuration can also be reached in a single time step. In the next section, these configurations will feature prominently in the mapping of several quantum error correction codes to the QDP architecture.

4.5 Error correction codes

In this section, we will apply the techniques we developed in the previous sections to map topological quantum error correction codes to the QDP.

4.5.1 Surface code

The planar surface code is well-studied to the point were we have an exact idea of how it should be implemented. In its rotated version, shown in Figure 1.2, one code patch contains $2d^2 - 1$ physical qubits encoding and maintaining a single logical qubit with distance d.Here, d^2 qubits are part of the memory and additional $d^2 - 1$ qubits are used for syndrome measurements. All of them are placed onto rhombus-shaped patch of square lattice, with ears at its boundaries. In the bulk, a checkerboard tiling of stabilizers, in which each plaquette engulf 5 qubits, is found – see Figure 4.7(a). Adjacent plaquettes share 2 memory qubits each, and each tile's central qubit is used for the measurement. In an error correction cycle, it collects the parity of the tile's memory qubits with CNOT-gates [80, 81, 96, 97], see Figure 4.7(b) and (c). Whether the parity is collected in the Hadamard or computational basis, meaning whether the stabilizer on those four qubits is $Z^{\otimes 4}$ or $X^{\otimes 4}$, is dependent on the shade of the plaquette on the checkerboard. In all figures of this thesis depicting surface code, we have chosen to distinguish Z-type stabilizers with a darker shade from white X-type stabilizers. Like in the previous chapter, we assume that gate operations can be performed in parallel as long as they do not share any resources. This leads the to a constant runtime for all parity collections. The measurement qubits, which now carry all syndrome information, are then read out. After being decoded, possible errors can be rectified and the error correction cycle concluded. Note that it is assumed that readout and correction are assumed to happen in single time steps, such that the entire circle has a runtime of $O(1)$.

Unfortunately, we cannot hope to run surface code cycles in the same manner on the QDP, not even if we neglect the issues of parallel operation and spurious crossings. As it turns out, our idea of how to run the code makes strong assumptions on the capabilities of the device that cannot be matched with the QDP: although CNOT-gates are possible to all adjacent qubits in the QDP, we have already argued to refrain from the use of CPHASE* gates for the sake of fidelity. This renders some of the two-qubit gates in Figure 4.7(b) and (c) nonlocal. Moreover, we require an additional qubit to be present in each stabilizer tile, to serve the measurement qubit as a reference in the syndrome extraction. Also, the readout procedure requires an empty dot along the barrier gate, which raises questions about the packing density of the qubits. To remedy all those issues, we present a revised version of the surface code cycles in Section 4.5.3.

4.5.2 2D color codes

Another important class of planar topological codes are the 2D color codes [83]. These codes are defined on 3-colorable tilings of the Euclidean plane. Two such tilings are featured in the so-called 6.6.6. and 4.8.8. codes, where hexagonal and square-octagonal shapes occur respectively. Similar to Figure 1.2, we can think of the memory qubits as sitting at the corners of those tiles, but the difference is here that every tile hosts two stabilizers, namely one in which X-operators are applied to their corners and another in which the same operators are replaced by Z. With suitable boundary conditions this construction encodes a single logical qubit with a distance d using an amount of d^2 physical qubits. See Figure 4.8 for examples of the 6.6.6. and 4.8.8. color codes of distance five, in which tiles and qubits are sketched. Note that, similar to Figure 1.2(b) and (c), these pictures do not include measurement qubits. Planar color codes have lower thresholds than the planar surface code but are more versatile when it comes to fault-tolerant gates, as the support the full Clifford

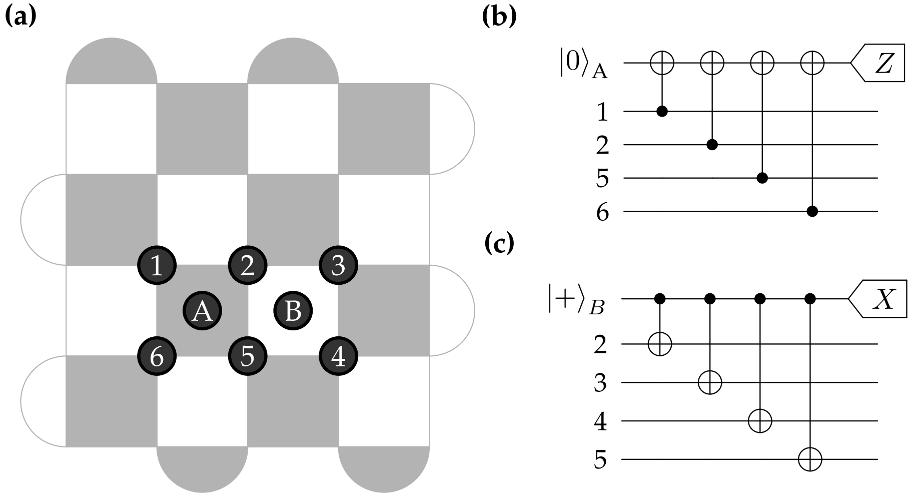


Figure 4.7. Stabilizer measurements in the surface code. **(a)** Distance-five code with some labeled qubits. Here, A and B label measurement qubits, while memory qubits carry numbers. The dark plaquettes indicate that the qubits at its corners are involved in a $X^{\otimes 4}$ stabilizer, where the syndrome is read out on the measurement qubit in its center. White plaquettes indicate regular parity measurements. **(b) & (c)** $Z-$ and X-stabilizer circuits [80, 81, 96, 97], with the qubits from panel (a).

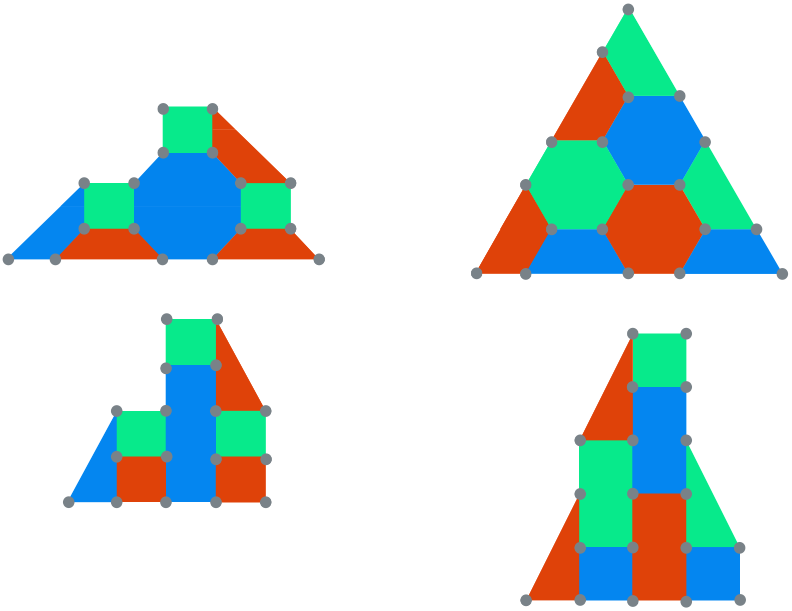


Figure 4.8. Distance 5 examples of the $4.8.8.$ (first from left) and $6.6.6.$ (third from left) color codes [83] and their deformed versions (second from left and fourth from left respectively). The vertices correspond to memory qubits and every colored face corresponds to both an X- and a Z-stabilizer to be measured. These stabilizers can be measured by using weight $4, 6$ and 8 versions of the circuits shown in Figure 4.7. The deformation of the codes does not change the code properties at all. They are a visual guide that facilitates the mapping the crossbar grid in Section 4.5.4.

group as a transversal set. In the next section we will focus on mapping these codes to the QDP using the concepts introduced in Section 4.4.

4.5.3 Surface code mapping

We now describe a protocol that maps the surface code on the architecture described in Section 4.3. The surface code layout has a straightforward mapping that places the memory qubits into even numbered columns, while X- and Z-measurement qubits can be found in the odd ones. This means we have single-qubit control over the set of all memory qubits and the set of all measurement qubits separately. We begin by changing the circuits performing the X- and Z-stabilizer measurements to work with $\sqrt{\text{SWAP}}$ rather than CNOT. We can emulate a

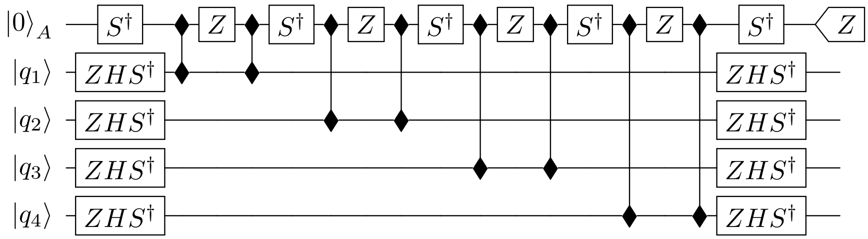


Figure 4.9. Z-stabilizer measurement circuit using the $\sqrt{\text{SWAP}}$ as the main two-qubit gate. The Z- and S-rotations can be performed by the timing procedure described in Section 4.3.3.3.

CNOT gate by using two $\sqrt{\text{SWAP}}$ gates interspersed with a Z-gate on the control plus some single qubit gates. As described in Section 4.3.3.5 the Z- and S-gates on the measurement qubit can performed by waiting, which means they can be performed locally while the single qubit operations on the memory qubits can be performed in parallel using the global unitary rotations described in Section 4.3.3.3. The X- and Z-circuits using $\sqrt{\text{SWAP}}$ are shown in Figure 4.9.

We will split up the quantum error correction cycle by first measuring all X-type stabilizers (the X-cycle) and then all Z-type stabilizers (Z-cycle). This means we can use the idle Z- (X-) measurement qubits as references for the X- (Z-) cycle measurements. For convenience we included a depiction of the surface code Z-cycle unit cell in Figure 4.10(right). Note that all panels in that figure depict the smallest possible building block of a code patch, not the patch itself. The qubit labeled 'A' is going to be measured in the Z-cycle. The numbered qubits are part of the memory and the qubit labeled 'B' is used as a reference for 'A' qubit. It is also the measurement qubit for the X-cycle. We now describe the steps needed to perform the Z-cycle in parallel on the entire surface code sheet. For convenience we ignore the surface code boundary conditions since these can be easily included. The X-cycle is equivalent up to different single qubit gates ($XS^\dagger$ instead of $ZHS^\dagger$ on the memory qubits, $HS^\dagger$ instead of $S^\dagger$ on the measurement qubits) and shifting every operation 2 steps up, e.g. setting $i \mapsto i+2$ in row indices.

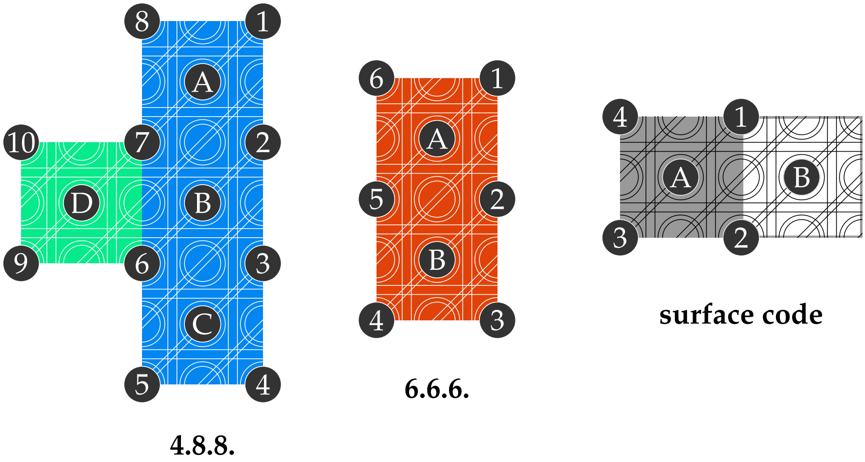


Figure 4.10. Unit cells of topological codes in the QDP. From left to right: deformed 4.8.8. color code, deformed 6.6.6. color code and surface code. Darkened circles correspond to qubits, where qubits used for measurement are labeled with letters, while memory qubits bear numbers. The shaded and colored plaquettes denote stabilizer tiles. Note that the depicted cells do not encode logical qubits, but are the smallest possible building blocks of a code patch. **4.8.8. code**: The qubit labeled 'A' is the is measured for the octagon (now a rectangle) stabilizer, while the qubit labeled 'D' has the same role for the square sub-cell. The qubit labeled 'B' is used to read out the qubit 'D' and the qubit labeled 'C' is used to read out the measurement qubit for the octagon cell directly below the square cell (not pictured). **6.6.6. code**: The qubit labeled 'A' is measurement qubit of that tile while the qubit labeled 'B' is used as a reference to read out the 'A' qubit for the unit cell directly to the bottom left (not pictured). **Surface code**: The qubit labeled 'A' is the measurement qubit of the for the Z-cycle stabilizer using 'B' as a reference. Their roles are reversed in the X-cycle.

The surface code Z-cycle

1. Initialize in the idle configuration.
2. Apply $ZHS^\dagger$ to all qubits in $\mathcal{R}$ (memory) and $S^\dagger$ to qubits in $\mathcal{B}$.
3. Go to right square configuration.
4. Go to rightward triangle configuration.
5. Perform CNOT between qubits A and 1 by performing VC[L] with
$$\mathrm{L} = \{(i,j) \;\|\; i \overset{2}{=} 1,\; j \overset{2}{=} 0,\; i+j \overset{4}{=} 3\}\,.$$
6. Perform CNOT between qubits A and 2 by performing VC[L] with
$$\mathrm{L} = \{(i,j) \;\|\; i \overset{2}{=} 0,\; j \overset{2}{=} 0, i+j \overset{4}{=} 2\}\,.$$
7. Go to idle configuration.
8. Go to left square configuration.
9. Go to leftward triangle configuration.
10. Perform CNOT between qubits A and 3 by performing VC[L] with
$$\mathrm{L} = \{(i,j) \;\|\; i \overset{2}{=} 1,\; j \overset{2}{=} 0,\; i+j \overset{4}{=} 1\}\,.$$
11. Perform CNOT between qubits A and 4 by performing VC[L] with
$$\mathrm{L} = \{(i,j) \;\|\; i \overset{2}{=} 0,\; j \overset{2}{=} 0,\; i+j \overset{4}{=} 0\}\,.$$
12. Go to idle configuration
13. Apply $ZHS^\dagger$ to all qubits in $\mathcal{R}$ and $S^\dagger$ to qubits in $\mathcal{B}$.
14. Apply measurement qubit correction step for qubit B as described in Section 4.4.1.1.
15. Go to measurement configuration.
16. Perform PSB measurement process as described in Section 4.4.3 using qubit B as reference to qubit A.
17. Go to idle configuration.

4.5.4 Color code mapping

The mapping of the color codes is largely analogous to that of the surface code. We begin with the 6.6.6. color code as it is easiest to map. First, the tiling on which the color code is must be deformed such that it is more amenable to the square grid structure of the QDP. This is fairly straightforward as can be seen from the $d = 5$ example in Figure 4.8. In the deformed tiling it is clear how to map the code to the QDP. We once again place all memory qubits in the even columns and all measurement qubits in the odd columns. This places the unit 'hexagon' seen in the deformed code into a patch of 3×5 dots on the QDP (see Figure 4.10 (right) for this unit tile). It also puts all memory qubits in $\mathcal{R}$ and 2 extra qubits into $\mathcal{B}$, both of which could be used as measurement qubit in the stabilizer circuit. We will always choose the top qubit ('A') of these two in the hexagon unit cell as the measurement qubit for the error correction cycles. The extra (bottom) qubit ('B') in the unit cell will be used as a reference for the unit hexagon to its direct left. This has the advantage of making the readout process independent of the measurement results of the previous cycles (as was the case in the surface code). Note also that all measurement qubits are positioned along diagonal lines on the QDP grid. This makes the quantum error correction cycle very analogous to the surface code. We once again must split up the X- and Z-cycles (again due to the limited single qubit rotations possible). Below we present the steps needed to perform the Z-cycle (which now measures a weight 6 operator). The X-cycle is identical up to differing single qubit rotations on the memory qubits.

The 6.6.6 color code Z-cycle

1. Apply Steps 1 to 11 in the surface code Z-cycle to perform CNOT gates between qubits A and the memory qubits $1, 2, 5, 6$ in the unit hexagon, ending in the idle configuration.

2. Go to idle configuration but with all even columns up and all odd columns down by performing VS[L] with

$$\mathrm{L} = \{(i, j, 1) \;\|\; i \overset{2}{=} 0,\ j \overset{2}{=} 0\} \cup \{(i, j, -1) \;\|\; i \overset{2}{=} 1,\ j \overset{2}{=} 1\}\,.$$

3. Go to right square configuration.
4. Go to rightward triangle configuration.
5. Perform CNOT between qubits A and 3 by performing VC[L] with

$$\mathrm{L} = \{(i,j) \;\|\; i \overset{2}{=} 1,\; j \overset{2}{=} 0,\; i+j \overset{4}{=} 1\}\,.$$

6. Go to idle configuration.
7. Go to left square configuration.
8. Go to leftward triangle configuration.
9. Perform CNOT by performing between qubits A and 4 by VC[L] with

$$\mathrm{L} = \{(i,j) \;\|\; i \overset{2}{=} 0,\; j \overset{2}{=} 0,\; i+j \overset{4}{=} 2\}\,.$$

10. Go to idle configuration.
11. Invert Step 6 by performing VS[L] with

$$\mathrm{L} = \{(i,j,-1) \;\|\; i \overset{2}{=} 0,\; j \overset{2}{=} 0\} \cup \{(i,j,1) \;\|\; i \overset{2}{=} 1,\; j \overset{2}{=} 1\}\,.$$

12. Apply $ZHS^{\dagger}$ to all qubits in $\mathcal{R}$ (memory) and $S^{\dagger}$ to qubits in $\mathcal{B}$.
13. Go to measurement configuration.
14. Perform PSB measurement process as described in Section 4.4.3 using qubit B as reference to A in the unit cell to the right.
15. Go to idle configuration.

Next up is the 4.8.8. color code. We deform the tiling on which the code is defined similarly to the 6.6.6. code. The deformed 4.8.8. code lattice can be seen in Figure 4.10 (left). We again place the memory qubits into $\mathcal{R}$ and the measurement qubits into the set $\mathcal{B}$. See Figure 4.10 for a layout of the unit cell of the 4.8.8. code on the QDP. Note that holds two different stabilizers. The square tile has one qubit (qubit 'D' in Figure 4.10) in $\mathcal{B}$, which we will use for the measurement. The deformed octagon tile has three qubits in $\mathcal{B}$. We will use the topmost qubit ('A') as the measurement qubit for the tile while the middle one (qubit 'B') serves as reference for the square tile measurement directly to its left. The bottommost qubit ('C') will be used to as a reference of the octagon directly below the square tile (not pictured). Because the structure of the 4.8.8. code is less amenable to direct mapping the stepping process is a little more involved. We will again only write down the Z-cycle with the X-cycle being the same up to initial and final single-qubit rotations on the memory qubits.

The 4.8.8 color code Z-cycle

1. Initialize in the idle configuration.
2. Apply $ZHS^\dagger$ to all qubits in $\mathcal{R}$ (memory) and $S^\dagger$ to qubits in $\mathcal{B}$.
3. Go to right square configuration.
4. Go to rightward triangle configuration.
5. Perform CNOT between qubits A and 1 and D and 7 by performing VC[L] with
$$\mathrm{L} = \{(i,j) \;\|\; i \overset{2}{=} 1,\; j \overset{2}{=} 0,\;\; i+j \overset{16}{=} 3 \vee 7\}\,.$$
6. Perform CNOT between qubits A and 2 as well as D and 6 by performing VC[L] with
$$\mathrm{L} = \{(i,j) \;\|\; i \overset{2}{=} 0,\; j \overset{2}{=} 0,\;\; i+j \overset{16}{=} 2 \vee 6\}\,.$$
7. Go to left square configuration.
8. Go to left triangle configuration.
9. Perform CNOT between qubits A and 8 and D and 9 by performing VC[L] with
$$\mathrm{L} = \{(i,j) \;\|\; i \overset{2}{=} 1,\; j \overset{2}{=} 0,\;\; i+j \overset{16}{=} 1 \vee 5\}\,.$$
10. Perform CNOT between qubits A and 7 and d and 10 by performing VC[L] with
$$\mathrm{L} = \{(i,j) \;\|\; i \overset{2}{=} 0,\; j \overset{2}{=} 0,\;\; i+j \overset{16}{=} 0 \vee 4\}\,.$$

11. Go to idle configuration.
12. Go to idle configuration but with all even columns up and all odd columns down by performing VS[L] with

$$\mathrm{L} = \{(i,j,1) \;\|\; i \overset{2}{=} 0,\; j \overset{2}{=} 0\} \cup \{(i,j,-1) \;\|\; i \overset{2}{=} 1,\; j \overset{2}{=} 1\}\,.$$

13. Go to right square configuration.
14. Go to rightward triangle configuration.
15. Perform CNOT between qubits A and 3 by performing VC[L] with

$$\mathrm{L} = \{(i,j) \;\|\; i \overset{2}{=} 1,\; j \overset{2}{=} 0,\; i+j \overset{16}{=} 3\}\,.$$

16. Perform CNOT between qubits A and 4 by performing VC[L] with

$$\mathrm{L} = \{(i,j) \;\|\; i \overset{2}{=} 0,\; j \overset{2}{=} 0,\; i+j \overset{16}{=} 2\}\,.$$

17. Go to idle configuration.
18. Go to left square configuration.
19. Go to leftward triangle configuration.
20. Perform CNOT between qubits A and 6 by performing VC[L] with

$$\mathrm{L} = \{(i,j) \;\|\; i \overset{2}{=} 1,\; j \overset{2}{=} 0,\; i+j \overset{16}{=} 1\}\,.$$

21. Perform CNOT between qubits A and 5 by performing VC[L] with

$$\mathrm{L} = \{(i,j) \;\|\; i \overset{2}{=} 0,\; j \overset{2}{=} 0,\; i+j \overset{16}{=} 0\}\,.$$

22. Go to idle configuration.
23. Invert Step 6 by performing VS[L] with

$$\mathrm{L} = \{(i,j,-1) \;\|\; i \overset{2}{=} 0,\; j \overset{2}{=} 0\} \cup \{(i,j,1) \;\|\; i \overset{2}{=} 1,\; j \overset{2}{=} 1\}\,.$$

24. Repeat Steps 2 - 23 but shifting $i \mapsto i+2$ and $j \mapsto j+1$.
25. Apply $ZHS^\dagger$ to all qubits in $\mathcal{R}$ and $S^\dagger$ to qubits in $\mathcal{B}$.
26. Go to measurement configuration.
27. Perform PSB measurement process as described in Section 4.4.3 using qubit B (unit cell to the right) as reference for qubit A and using qubit C as reference for qubit D.
28. Go to idle configuration.

4.6 Discussion

In this section, we evaluate the mapping of the error correction codes described above and argue numerically that it is possible to attain the error suppression needed for practical universal quantum computing. We will do this exercise for the planar surface code, as it is the most popular and best understood error correction code. The description given in Section 4.5.3 assumes that all operations can be implemented perfectly in parallel. In practice though, for the reasons outlined in Section 4.4 many operations that can in principle be done in parallel will be done in a line-by-line fashion. Note that for surface code in an array like this, the side lengths of a quadratic grid scale linearly with the code distance as $N = 2d + 1$. This means that the time performing a surface code cycle (and thus the number of errors affecting a logical qubit) rises linearly with the code distance and hence this mapping of the surface code will not exhibit an error correction threshold. As a consequence, the error probability of the encoded qubit (the logical error probability) cannot be made arbitrarily small but rather will exhibit a minimum for some particular code distance after which it will start rising with increasing code distance. Also, the code distance characterizing the minimum will depend nontrivially on the error probability of the code qubits. This is not a very satisfactory situation from a theoretical point of view, but being pragmatic we are not so much interested in asymptotic statements but rather in whether the logical error probability can be made small enough to allow for realistic computation [97]. As a target logical error probability we choose $P_L = 10^{-20}$ as at this point the computation is essentially error free (for comparison, a modern classical processor has an error probability around 10^{-19} [84]). We will use this number as a benchmark to assess if and for what error parameters the surface code mapping in the QDP yields a "practical" logical qubit. In order to assess this we must consider in more detail the sources of error afflicting the surface code operation on the QDP. We will begin by detailing how the surface code is likely to be implemented in practice on the QDP and afterwards consider how this impacts the error behavior of the logical surface code qubit. We will distinguish two classes of error sources: operation induced errors and decoherence induced errors.

4.6.1 Practical implementation of the surface code

Here we present an mapping of the surface code based on the one presented in Section 4.5.3 but differing in the amount of time-steps used to perform certain operations. In particular, we choose to do all shuttle and two-qubit-gate operations in a line-by-line manner. This is a specific choice which we expect will work well but variations of this protocol are certainly possible. As mentioned above, this will mean that the time an error correction cycle takes will scale with the code distance. This means it is important to keep track of the time needed to perform a cycle. We will do this while describing line-by-line operation of the surface code cycle in greater detail below.

In practice, we will perform the protocol in Section 4.5.3 in the following manner. We begin by performing Step 1 and 2 for all qubits. Then we apply Steps $3-7$ but only in rows 0 and 1. Note that after performing these steps on only the first two columns we are back in the *idle* configuration. Now we repeat the previous for rows 2 and 3 and so forth until we reach the end of the grid. Having done these operations we are at the end of Step 7 (go to *idle* configuration) and the grid is the *idle* configuration. We now repeat the same process to perform Step $8-12$ of Section 4.5.3. Next we perform Step 13 which can be done globally. Hereafter we perform step 14 (measurement qubit correction) in standard line-by-line fashion. Note that even in an ideal implementation Step 14 has to be done line-by-line in the worst case. After this we perform Step 15 (go to *measurement* configuration) in a line-by line manner and similarly for Steps 16 (PSB/readout procedure) and 17 (go to *idle* configuration).

Note that in this line-by-line implementation there is a slight asymmetry between the X- and Z-cycles. In Table 4.4, we count the number of time steps that accumulate for every operation type in each program step in Section 4.5.3. We also calculate the number of time steps (per operation type) needed for the full surface code error correction cycle.

4.6.2 Decoherence induced errors

Decoherence induced errors are introduced into the computation by uncontrolled physical processes in the underlying system. The effect of these processes is called decoherence. Decoherence happens even if a

Step	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$\sqrt{\text{SWAP}}$ gates					$2d$	$2d$				$2d$	$2d$						
Z-rotations					$2d$	$2d$				$2d$	d						
Shuttlings			d	d			d	d	d			d			$5d$	$2d$	$3d$
Global rotations		1											1		1		
Measurements																d	

	Average	Total
$\sqrt{\text{SWAP}}$ gates	$8d$	$16d$
Z-rotations	$7d$	$14d$
Shuttlings	$16d$	$32d$
Global rotations	3	6
Measurements	d	$2d$

Table 4.4. Time-step count per operation type and program step for the line-by-line implementation of the surface code cycle described in Section 4.5.3. The number of time-steps is quoted in terms of the code distance d. This table does not specify the exact order in which the operations happen, see Section 4.6.1 for an explanation of the time flow. Note that the table shows the average of the time-step counts for the X- and Z-cycles. The actual count for the individual X- and Z-cycles is slightly different due to the boundary conditions of the surface code. Table cells that are left empty signify zero entries.

qubit is not being operated upon and the amount of decoherence happening during a computation scales with the time that computation takes. Therefore, to account for decoherence induced errors during the error correction cycle we need to compute how long an error correction cycle takes. Generally any operation on the QDP takes a certain amount of time denoted by τ. We distinguish again five different operations: (sw) two-qubit $\sqrt{\text{SWAP}}$ gates, (sh) qubit shuttle operations, (z) single qubit Z-gates by waiting, (gl) global single qubit operations and (m) qubit measurements. The time they take we will denote by $\tau_{sw}, \tau_{sh}, \tau_z, \tau_{gl}$ and τ_m respectively. In Table 4.4 we count the total time taken by the surface code error correction cycle using the mapping described in Sections 4.5.3 and 4.6.1. Table 4.5 summarizes the total number of time-steps for every gate type for a full surface code error correction cycle. Following that table, the total time $\tau_{\text{total}}(d)$ as a function of the code distance d is given by

$$\tau_{\text{total}}(d) = 16d\,\tau_{sw} + 32d\,\tau_{sh} + 14d\,\tau_z + 6\,\tau_{gl} + 2d\,\tau_m. \tag{4.14}$$

This total time can be connected to an error probability by invoking the mean decoherence time of the qubits in the system, the so called T_2 time [88,

Symbol	Operation	Time-steps per cycle
τ_{sw}	$\sqrt{\text{SWAP}}$ gates	$16d$
τ_{sh}	Shuttlings	$32d$
τ_z	Z rotations by waiting	$14d$
τ_{gl}	Global qubit rotations	6
τ_m	Measurements	$2d$

Table 4.5. Time steps required for one error correction cycle of surface code.

98]. We neglect the influence of T_1 in this calculation as it is typically much larger than T_2 in silicon spin qubits [18, 99]). We can find the decoherence induced error probability P_{dec} [88, Page 384] as

$$P_{dec}(d) = \frac{\tau_{\text{total}}(d)}{2T_2}. \tag{4.15}$$

Next we investigate operation induced errors. These will typically be larger than decoherence induced errors but will not scale with the distance of the code.

4.6.3 Operation induced errors

Operation induced errors are caused by imperfect application of quantum operations to the qubit states. According to the five types of operations, we will denote the probability of an error afflicting them by $P_{sw}, P_{sh}, P_z, P_{gl}$ and P_m respectively. In Table 4.6 we list the total number of operations of a given type that memory and measurement qubits participate in over the course of a surface code cycle. In Section 4.8 we give a more detailed per-step overview of the operations performed on memory and measurement qubits. For clarity we have chosen qubit 1 in Figure 4.10 (right) as a representative of the memory qubits and qubit A as a representative of the measurement qubits. Other qubits in the code might have a different ordering of operations but their counts will be the same, except for the qubits located at the boundary of the code patch for which the given counts are an upper bound. For each operation we also calculate the average number of this times it involves memory and measurement qubits. This average number will serve as our measure of operationally induced error.

	Memory qubit			Z-measurement qubit			Average
	Z-cycle	X-cycle	Total	Z-cycle	X-cycle	Total	
$\sqrt{\text{SWAP}}$ gates	4	4	8	8	0	8	8
Z-rotations	0	0	0	7	0	7	3.5
Shuttlings	2	4	6	10	4	14	10
Global rotations	2	2	4	2	3	5	4.5
Measurements	0	0	0	1	1	2	1

Table 4.6. This table lists the total number of operations per qubit type, over the course of a surface code cycle. In Section 4.8 we give a more detailed per-step overview of the operations performed. For clarity we have chosen qubit 1 in Figure 4.10 (right) as a representative of the memory qubits and qubit A as representative of the measurement qubits..

4.6.4 Surface code logical error probability

By tallying up the contributions from operational and decoherence induced errors we can construct a measure for the total error probability per error correction cycle experienced by all physical qubits that make up the code. Note that this a rather crude model that disregards possible influences from inter-qubit correlated errors and time-like correlated errors. Nevertheless it serves as a useful first approximation to the performance of the surface code on the QDP. We define the average per qubit per cycle error probability P_{tot} as

$$P_{\text{tot}}(d) = 8P_{sw} + 3.5P_{sh} + 10P_z + 4.5P_{gl} + P_m + P_{dec}(d). \tag{4.16}$$

Note that this quantity depends linearly on the code distance d. We can plug this total per cycle error probability P_{tot} into an empirical equation for the logical error probability P_L derived in [97]:

$$P_L = 0.03\left(\frac{P_{tot}(d)}{8P_{th}}\right)^{\frac{d+1}{2}}, \tag{4.17}$$

where P_{th} is the per-step fault-tolerance threshold of the surface code, which we take to be $P_{th} = 0.0057$ following the result in [97]. The factor of 8 is inserted to account for the fact that the empirical relation derived in [97] is between the physical *per-step* error rate and the logical *per cycle* error rate and the protocol analyzed in [97] requires 8 time-steps per surface code error correction cycle. This is an approximation but it will serve our purposes of getting a basic initial estimate of the logical error

Operation	Error probability	Time
Two-qubit $\sqrt{\text{SWAP}}$ gate	$P_{sw} = 10^{-3}$	$\tau_{sw} = 20\text{ns}$
Coherent shuttle	$P_{sh} = 10^{-3}$	$\tau_{sh} = 10\text{ns}$
Z-rotation by waiting	$P_z = 10^{-3}$	$\tau_z = 100\text{ns}$
Global qubit rotation	$P_{gl} = 10^{-3}$	$\tau_{gl} = 1000\text{ns}$
Measurement	$P_m = 10^{-3}$	$\tau_m = 100\text{ns}$

Table 4.7. Error probabilities and times for the five elementary operations of the QDP.

rate. In Table 4.7, we quote error probabilities and operation times that will be plugged into (4.16). These numbers are projections from [18] and references therein. To convert the operation times into decoherence induced error we use the estimated T_2 time of quantum dot spin qubits in ^{28}Si quoted as $T_2 = 10^9\text{ns}$ [18, 99] and (4.15). Plugging these numbers into (4.16) we get the following linear function of the code distance

$$P_{tot} = 2.7 \times 10^{-2} + 2.8d \times 10^{-5} \tag{4.18}$$

which we can plug into the empirical model (4.17). In Figure 4.11 we plot the logical error probability P_L versus code distance. Note that for the experimental numbers provided the practical quantum computing benchmarking $\log(P_L) = -20$ is reached for a code distance of $d = 37$. The maximal code distance for the experimental parameters is $d = 155$ for which the logarithmical logical error probability reaches $\log(P_L) = -41$, after which it starts increasing again. We also plot what would happen if we had the power to operate the QDP (with quoted device parameters) completely in parallel. The physical error rate of the latter scenario is calculated setting $d = 1$ in (4.18). Note that the difference between parallel and crossbar style operation is not that big, the parallel version reaches $P_L = 10^{-20}$ for $d = 31$. This rough model provides some quantitative justification for the implementation of planar error correction codes in the QDP even in the absence of the ability to arbitrarily suppress logical errors. Note also that, due to the long coherence times of the QDP spin qubits [18, 99], the dominant terms in the expression for the total error probability P_{tot} are those associated with operation induced errors. This provides justification for the line-by-line application of two-qubit gates discussed in Section 4.4.2, which takes a longer time to perform but improves gate quality. It also means that long coherence times and/or fast operation times are likely critical to the success of a crossbar based scheme. This concludes our discussion of the QDP mapping of the surface code. A similar exercise can be done for the 6.6.6. and 4.8.8. color codes but due to their lower thresholds [100], the results will likely be less positive for current experimental parameters.

4.7 Conclusion

We analyzed the architecture presented in [18], focusing on its crossbar control system. Building on this analysis we presented procedures for mapping the planar surface code and the 6.6.6. and 4.8.8. color codes. Because the line-by-line operation of the crossbar architecture means the noise in a single error correction cycle scales with the distance, it is not possible to arbitrarily suppress the logical error rate by increasing the code distance. Instead there will be some "optimal" code distance for which the logical error rate is the lowest. Using numbers for [18] and an empirical model taken from [97] we analyzed the logical error behavior of the surface code mapping and found that, for current experimental numbers, it appears plausible to achieve logical error probabilities below $P_{log} = 10^{-20}$, making practical quantum computation possible. However, we strongly stress that this is a rather crude estimate and a more detailed answer would have to take into account the details of the dominant error processes in quantum dot qubits. It must also take into account that while it is possible to achieve certain low noise gates and good coherence times in quantum dots qubits in isolation this does not necessarily mean they will be practically achievable in the current QDP design. A future research direction would be to perform much more detailed simulations of this crossbar system, perhaps with input from future experiments. In such a simulation the effect of correlated errors (which might feasibly appear in a crossbar architecture) could be investigated.

Another possible research direction would be to use the currently developed machinery to map more exotic quantum error correction codes. A first step in this direction would be the implementation of variants of the surface code with more resistance to biased noise [101, 102]. Due to the possibility of qubit shuttling, also codes with long distance stabilizers could in principle be implemented. Codes such as the 3D gauge color codes might be prime candidates for this kind of treatment. How-

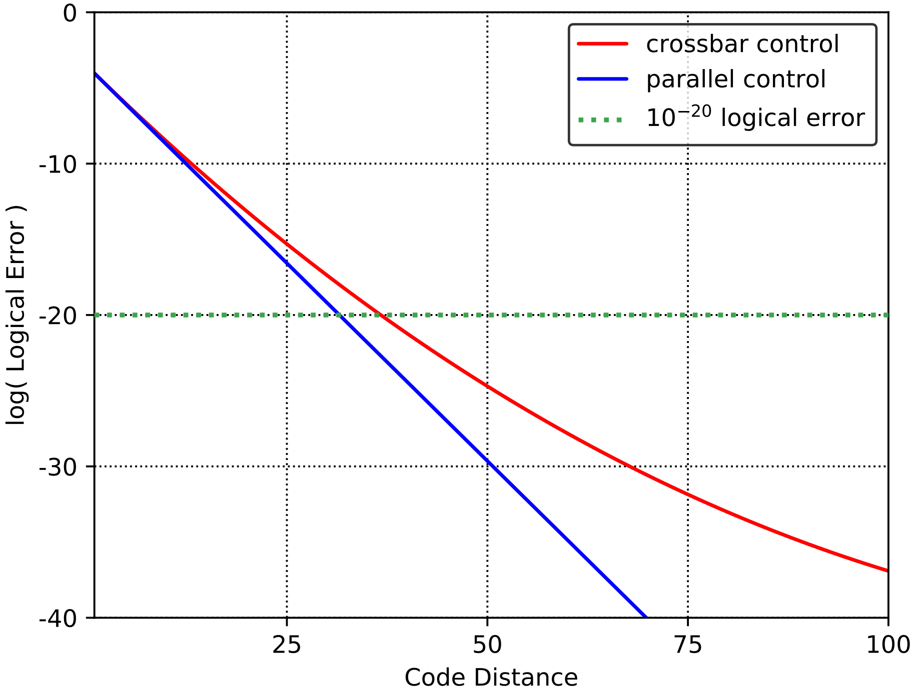


Figure 4.11. Plot of logical error probability versus code distance for the empirical model given in (4.17) with experimental parameters given in Table 4.7. Note that the logical error probability for crossbar operation goes below $P_L = 10^{-20}$ for $d = 37$. This is only slightly slower that parallel operation, which reaches $P_L = 10^{-20}$ for $d = 31$. Due to the scaling of crossbar operation with the code distance the logical error probability bottoms out at some point. This however does not happen until $d = 155$ (not shown) for a logical error rate of $P_L = 10^{-41}$, which is not practically relevant. This rough model gives good indication it is possible to create very low logical error surface code logical qubits in the QDP.

ever, barring some special cases, parallel shuttling is currently being performed in a line-by-line manner. A general classical algorithm for generating optimal (in time) shuttling-steps from an initial to a final BOARDSTATE would vastly simplify the task of mapping more exotic codes and also general quantum circuits. Such an algorithm would probably be useful for any future crossbar quantum architecture.

Lastly, there are important aspects of quantum error correction that are not discussed in this paper. Two of these aspects are the ability to store multiple logical qubits simultaneously and the ability to perform quantum operations on the logical qubits. A popular way of performing these tasks is by encoding multiple logical qubits in a single surface code sheet by introducing topological defects in to the surface code sheet [97]. This process involves not measuring stabilizers at certain points in the sheet, thus creating extra degrees of freedom which can store logical information. The code distance of the code is given by the physical distance (measured in number of physical qubits) between the defects. Operations can then be performed on these logical qubits by moving the defects around each other, a process known as braiding. We think this approach is not natural to the constraints of the crossbar architecture for the following reasons

- Encoding qubits as defects would mean the size of the surface code sheet would scale as the number of encoded qubits. Hence also, in our implementation, the physical error probability per QEC cycle would scale with the number of qubits. This would put an upper limit on the number of qubits that can be implemented.
- Creating and moving defects around requires turning on and off measurements for certain stabilizers in a local manner. This locality runs counter to the design ideas of the crossbar architecture.
- Given that the size of the surface code sheet would scale with the number of logical qubits one would likely face significant issues involving uniformity of control parameters of the entire sheet. This would be a significant issue even if the scaling of the physical error probability can be avoided by clever implementation.

However, we can envision a mode of computation that we speculate is more amenable to this architecture by thinking of an architecture composed of separate modules containing a single logical qubit. We refer to Figure 7 of [18] for a proposal of implementation. Inside each module our surface code protocol could be run with the ideal code distance given physical error parameters setting the size of these modules. We could then perform logical X- and Z-gates transversally within the modules and we could perform CNOT gates between adjacent modules via lattice surgery. Note that lattice surgery, which involves the turning on and off of stabilizer patches in regular patterns (see [103] for an introduction to lattice surgery), is very amenable to the constraints of the architecture, implying that a high degree of parallelization could be achieved when mapping lattice surgery techniques to the QDP.

4.8 Supplement: surface code operation counts

(The reader may find the corresponding tables on the next pages.)

Steps	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	Z-cycle Total
$\sqrt{\text{SWAP}}$ gates					2	2				2	2							8
Z-rotations					2	2				2	1							7
Shuttlings			1				1	1				1		2	1	2	1	10
Global rotations		1											1					2
Measurements																1		1

Table 4.8. Operation counts per step for the Z-measurement during the Z-cycle of the surface code described in Section 4.5.3 and Section 4.6.1. Specifically the Z-measurement qubit is taken to be qubit A in Figure 4.10(right). Table cells that are left empty signify zero entries.

Steps	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	X-cycle Total
$\sqrt{\text{SWAP}}$ gates																		0
Z-rotation																		0
Shuttlings														2	1		1	4
Global rotations		1											1		1			3
Measurements																1		1

Table 4.9. Operation counts per step for the Z-measurement qubit during the X-cycle of the surface code described in Section 4.5.3 and Section 4.6.1. The Z-measurement qubit is taken to be qubit A in Figure 4.10(right). Table cells that are left empty signify zero entries.

Steps	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	Z-cycle Total
$\sqrt{\text{SWAP}}$ gates					2						2							4
Z-rotations																		0
Shuttlings			1	1														2
Global rotations		1											1					2
Measurements																		0

Table 4.10. Operation counts per step for a memory qubit during the Z-cycle of the surface code described in Section 4.5.3 and Section 4.6.1. Specifically the memory qubit is taken to be qubit 1 in Figure 4.10(right) but other memory qubits will have the same gate count up to a possible reordering of steps. Table cells that are left empty signify zero entries.

Steps	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	X-cycle Total
$\sqrt{\text{SWAP}}$ gates						2						2						4
Z-rotations																		0
Shuttlings								1	1						1*		1*	4
Global rotations		1											1					2
Measurements																		0

Table 4.11. Operation counts per step for a memory qubit during the X-cycle of the surface code described in Section 4.5.3 and Section 4.6.1. Specifically the memory qubit is taken to be qubit 1 in Figure 4.10(right) but other memory qubits will have the same gate count up to a possible reordering of steps. Table cells that are left empty signify zero entries.

*: Only half of the memory qubits move during this step. In the total operation count this shuttling is counted towards all memory qubits.

4.9 Notations

$a \overset{b}{=} c$	Shorthand for $a \bmod b \ = \ c$.
$[a : b]$	Set of integers from a to b.
(i, j)	Dot locations, in row i and column j.
BOARDSTATE	$(N \times N)$ Matrix mirroring the charge distribution over the grid of dots.
$\mathcal{B}$	Qubits in grid columns of high magnetic field.
CNOT	Controlled-Not gate: $\vert 0\rangle\langle 0\vert \otimes \mathbb{I} + \vert 1\rangle\langle 1\vert \otimes X$.
d	Distance of a quantum code.
CPHASE*	Effective controlled-phase gate, see (4.3).
D$[i][t]$	Set diagonal line i to potential level t, see Table 4.1.
H	Hadamard gate: $(X + Z)/\sqrt{2}$.
H$[i]$	Pulsing horizontal barrier i, see Table 4.1.
HC$[i, j]$	CNOT-gate along horizontal direction, Table 4.3 and Figure 4.4.
HI$[i, j]$	OPCODE for CPHASE* gate between qubits (i, j) and $(i,\ j + 1)$, see Table 4.3.
HS$[i, j, k]$	Horizontal shuttling at (i, j) with flow k, see Table 4.2.
M$[i, j, k]$	Measuring the qubit at (i, j) with the qubit at $(i,\ j+k)$ as a reference, see Table 4.2.
N	The grid inside the processor has the size of $N \times N$ quantum dots, see Figure 4.1(a).
PSB	Pauli spin blockade.
QDP	Abbreviation for *Quantum Dot Processor*, the term we use to describe the proposed quantum device.
$\mathcal{R}$	Qubits in grid columns with low magnetic field.
S	Square-root of a Pauli-Z gate: $\vert 0\rangle\langle 0\vert + i\,\vert 1\rangle\langle 1\vert$.
$\sqrt{\text{SWAP}}$	Square-root of swap gate, (4.2).
V$[i]$	Pulse vertical barrier gate i, see Table 4.1.

$\mathrm{VC}[i, j]$	CNOT-gate in vertical direction, see Table 4.3 and Figure 4.4.
$\mathrm{VI}[i, j]$	OPCODE for $\sqrt{\text{SWAP}}$ gate between qubits (i, j) and $(i + 1,\ j)$, see 4.3.
$\mathrm{VS}[i, j, k]$	Vertical shuttling at (i, j) with flow k, see Table 4.2.

Publications and Preprints

1. **(Chapter 3)** M. Steudtner and S. Wehner. *Quantum codes for quantum simulation of fermions on a square lattice of qubits.* Phys. Rev. A **99**, 022308 (2019).

2. **(Chapter 2)** M. Steudtner and S. Wehner. *Fermion-to-qubit mappings with varying resource requirements for quantum simulation.* New J. Phys. **20**, 063010 (2018).

3. **(Chapter 4)** J. Helsen, M. Steudtner, M. Veldhorst, and S. Wehner. *Quantum error correction in crossbar architectures.* Quantum Sci. Technol. **3**, 035005 (2018).

4. **(Chapter 4)** R. Li, L. Petit, D. P. Franke, J. P. Dehollain, J. Helsen, M. Steudtner, N. K. Thomas, Z. R. Yoscovits, K. J. Singh, S. Wehner, L. M. K. Vandersypen, J. S. Clarke, and M. Veldhorst. *A crossbar network for silicon quantum dot qubits.* Sci. Adv. **4**, eaar3960 (2018).

5. J. R. McClean, K. J. Sung, I. D. Kivlichan, Y. Cao, C. Dai, E. S. Fried, C. Gidney, B. Gimby, P. Gokhale, T. Häner, T. Hardikar, V. Havlíček, O. Higgott, C. Huang, J. Izaac, Z. Jiang, X. Liu, S. McArdle, M. Neeley, T. O'Brien, B. O'Gorman, I. Ozfidan, M. D. Radin, J. Romero, N. Rubin, N. P. D. Sawaya, K. Setia, S. Sim, D. S. Steiger, M. Steudtner, Q. Sun, W. Sun, D.Wang, F. Zhang, and R. Babbush. *OpenFermion: the electronic structure package for quantum computers.* arXiv:1710.07629 (2017).

6. E. C. Andrade, M. Steudtner, and M. Vojta. *Anderson localization and momentum-space entanglement.* J. Stat. Mech.: Theory Exp. **2014.7**, P07022 (2014).

Summary

This thesis is a collection of theoretical works aiming at adjusting quantum algorithms to the hardware of quantum computers. The overarching topic of these efforts is to enable digital quantum simulation, the process of approximating the ground state of an arbitrary physical system with elementary operations of a quantum computer. For fermionic systems, a class including molecules and materials, the impact of quantum computing would be undoubtedly high, and algorithms exist for their simulation. However, there is a certain gap between the requirements of those algorithms and what actual quantum devices can provide: it seems that our expectations of a fully-fledged quantum computer still exceed our capabilities to build it. To make quantum simulation feasible, we seek to adapt quantum algorithms to three different types of device limitations within this thesis. In particular, we address two of those issues in the context of fermionic quantum simulation and discuss the third while doing quantum error correction (which is a low-level task of a quantum computer).

Firstly, we consider a shortage of quantum bits. Qubits are a valuable resource that is squandered when quantum information is stored uncompressed in a quantum memory. This is in particular the case with digital quantum simulation when the physical system is modeled using the Jordan-Wigner transform – a longstanding standard of representing fermionic systems as spins (equivalent to qubits). Here, only a small number of configurations that the quantum memory could encode are actually relevant for the simulation. In chapter two, we therefore set out to save qubits by restricting ourselves to only those relevant configurations. The key to our method is that all quantum data stored in the qubit memory is a superposition of bit strings describing the simulated system. In the Jordan-Wigner case, these bit strings are readable as fermionic oc-

cupations (different states describing how the fermions occupy their host system). By using binary codes, only physically relevant occupations are mapped to somewhat shorter bit strings, and so stored on fewer qubits. This procedure however comes at the expense of the simulation runtime, as the fermionic interactions are mapped to operations more complicated than in the Jordan-Wigner case – an effect that increases the stronger the information is abstracted. Nonetheless, our efforts result in a scalable method that helps reducing the qubit requirements a bit.

A different limitation comes from the connectivity of qubits in a quantum device, and the need to keep quantum algorithms short. Of course, we will not be able to perform arbitrary quantum operations on our memory. Qubits have physical locations, and our ability to perform operations that couple two of them is geometrically limited. While scientists and engineers work on realizing devices with square lattice connectivity, fermionic quantum simulation is unfortunately not harnessing that connectivity in an optimal way. This problem is related to the locality of fermionic interactions: when simulating a term that is geometrically local in the fermionic system's two-dimensional embedding, one would like to only read and write on a local group of qubits. In this way, the quantum algorithm can be parallelized. Considering the limits set by decoherence, the resulting decrease in algorithmic runtime might even be the deciding factor for the feasibility of a computation. In chapter three, we again tinker with the way physical states are represented in the memory. The Jordan-Wigner transform, which is not locality-preserving, is here again concatenated with a code layer. This time however, the code is quantum – a construction with code words that can no longer be thought of as bit strings. Rather than relaxing the resource requirements, the number of qubits almost needs to be doubled. This increase in memory is necessary to store nonlocal interactions locally on single qubits, from where those interactions can then be retrieved via local operations. Not only does this allow to simulate two-dimensional systems in a constant number of algorithmic steps, but ensures parallelizability in general.

Lastly, we consider a limitation that is swept under the rug fairly often. For devices with many qubits, constraints on parallel operation will appear due to a limited capability to address the involved qubits at the same time. A realistic layout for a future quantum device is a densely-packed matrix of qubits, so dense in fact that they can only be manipulated by wires contacting the matrix periphery. In chapter four, we consider such a crossbar design for spin qubits in silicon quantum dots. There, the pulsing of lines that go into the chip allows us to set the electric dot potentials and interdot confinement barriers, by which quantum operations can be facilitated. For all those operations at least two lines must be pulsed affecting only the part of the chip at which they intersect. However, when attempting to run several operations in parallel, there are not just the crosspoints that we intend, but also others that would not occur if the operations were performed in sequence. At those spurious crossings, unanticipated quantum operations are induced. As a way out, we use the quirks of the proposed architecture and for instance move electrons (qubits) between adjacent dots: when the dots at the spurious crossings are empty, spurious operations cannot corrupt the quantum memory. By providing charge distributions with these properties, as well as instructions for the operation of the lines, we find quantum error correction programs compatible with the crossbar architecture.

Samenvatting

Dit proefschrift bevat een verzameling theoretische studies gericht op het aanpassen van quantumalgoritmen aan de hardware van quantumcomputers. Het overkoepelende onderwerp van dit onderzoek is om digitale quantumsimulatie mogelijk te maken, het proces van het benaderen van de grondtoestand van een willekeurig fysisch systeem met behulp van een quantumcomputer. De impact van een quantumcomputer kan potentieel heel groot zijn voor de simulatie van eigenschappen van moleculen en materialen. Er bestaan algoritmen voor de simulatie van elektronische eigenschappen, maar die zijn niet zonder meer toepasbaar op realistische quantumschakelingen. Het lijkt erop dat onze verwachtingen van een volwaardige quantumcomputer nog steeds onze mogelijkheden overtreffen om er zo één te bouwen. Om quantumsimulatie toch mogelijk te maken, proberen we in dit proefschrift quantumalgoritmen aan te passen aan drie verschillende soorten realistische beperkingen. In het bijzonder behandelen we twee van die beperkingen in de context van fermionische quantumsimulatie en bespreken we de derde in verband met quantumfoutcorrectie.

De eerste beperking is het tekort aan quantumbits. Omdat bestaande quantumcomputers hooguit enkele tientallen qubits bevatten, moeten we quantuminformatie op een efficiënte manier opslaan. Veel algoritmen zijn niet efficiënt, en dit is met name het geval bij digitale quantumsimulatie wanneer het fysieke systeem wordt gemodelleerd met behulp van de Jordan-Wigner-transformatie – een bekende manier om elektronische orbitalen te vervangen door een spinketen. Deze manier om informatie op te slaan verspilt qubits, omdat slechts een kleine hoeveelheid van de spinconfiguraties relevant zijn voor de simulatie van de elektronen. In hoofdstuk twee willen we daarom qubits sparen door ons te beperken tot alleen de relevante configuraties. De sleutel tot onze methode is dat alle quantumgegevens die zijn opgeslagen in het qubit-geheugen een superpositie zijn van bitstrings die het gesimuleerde systeem beschrijven. Door het gebruik van binaire codes worden alleen fysisch relevante superposities toegewezen aan iets kortere bitstrings, en dus opgeslagen in minder qubits. Deze procedure gaat weliswaar ten koste van een langere duur van de simulatie, maar dat nadeel valt in het niet bij het voordeel dat de simulatie kan worden uitgevoerd met een kleiner aantal qubits.

De tweede beperking komt van de connectiviteit van qubits. Qubits hebben vaste posities en ons vermogen om bewerkingen uit te voeren is doorgaans beperkt tot nabijgelegen qubits. De meeste quantumcomputers worden op een tweedimensionaal vierkant rooster ontworpen, waarbij elke qubit gekoppeld kan worden aan de vier buren. De bestaande algoritmes van quantumsimulatie maken geen optimaal gebruik van deze connectiviteit, doordat de lokaliteit van interacties in de simulatie verloren gaat. Het is van essentieel belang dat lokale interacties tussen elektronen geïmplementeerd kunnen worden in een lokale groep van nabijgelegen qubits. Op deze manier kan het quantumalgoritme parallel worden geschakeld en blijft de looptijd kort. Gezien de limieten die zijn vastgesteld door decoherentie, kan de resulterende afname in algoritmische looptijd zelfs de beslissende factor zijn voor de haalbaarheid van een berekening. In hoofdstuk drie sleutelen we opnieuw aan de manier waarop fysische toestanden in het geheugen worden weergegeven. De Jordan-Wigner-transformatie, die de lokaliteit niet behoudt, wordt hier samengevoegd met een codelaag. Deze keer is de code echter quantum – een constructie met codewoorden die niet langer als bitstrings kunnen worden beschouwd. Om de lokaliteit te behouden betalen we een prijs, het aantal qubits moet verdubbeld worden. Maar het grote voordeel is dat het nu mogelijk wordt om tweedimensionale systemen in een constant aantal algoritmische stappen te simuleren.

Ten slotte beschouwen we de derde beperking, die vrij vaak onder het tapijt wordt geveegd. Voor apparaten met een groot aantal qubits zullen beperkingen voor parallelle werking verschijnen vanwege een beperkte mogelijkheid om al de qubits tegelijkertijd aan te sturen. Een realistische lay-out voor een toekomstig quantumapparaat is een dicht opeengepakte matrix van qubits, zo dicht zelfs dat ze worden gemanipuleerd door draden die contact maken met de periferie van de matrix. In hoofdstuk vier beschouwen we een dergelijk dwarsbalkontwerp voor spinqubits in quantumdots van silicium. De qubits worden aangestuurd door spanningslijnen, en voor quantumbewerkingen moeten ten minste twee lijnen worden geactiveerd die alleen het deel van de chip beïnvloeden waarop ze elkaar snijden. Wanneer we echter proberen meerdere bewerkingen parallel uit te voeren, worden niet alleen de kruispunten die we wensen geactiveerd, maar ook andere kruispunten waar dit niet zou moeten. Bij die valse kruisingen worden ongewenste quantumoperaties geïnduceerd. Als uitweg gebruiken we een bijzondere eigenschap van de voorgestelde architectuur en verplaatsen we bijvoorbeeld elektronen (qubits) tussen aangrenzende quantumdots: wanneer de dots aan de onbedoelde kruisingen leeg zijn, kunnen ongewenste operaties het quantumgeheugen niet beschadigen. Door het aanbieden van ladingsverdelingen met deze eigenschappen, evenals instructies voor de werking van de lijnen, vinden we quantumfoutcorrectieprogramma's die compatibel zijn met de dwarsbalkarchitectuur.

Curriculum vitae

I was born in 1991 in Löbau, Germany, where I grew up and received my school education. In 2010, after having been granted the entrance qualification for higher education by my local high school 'Geschwister-Scholl-Gymnasium', I enrolled at Dresden University of Technology in the subject of physics. My undergraduate studies, that covered a broad spectrum of topics, were complemented by an internship at the Helmholtz research facility in Dresden-Rossendorf in 2011. In 2013, my interest in theoretical physics had grown, and so I obtained my Bachelor degree in the solid state theory group of Matthias Vojta. With my course work focusing on condensed matter theory, I returned to Prof. Vojta's group one year later, writing my Master thesis on a topic in quantum magnetism. Towards the end of 2015, I received my Master of Science degree and was hired by Carlo Beenakker to join his group in Leiden as a PhD student in early 2016. As soon as I joined the Beenakker group, I engaged in collaborative efforts with Stephanie Wehner, and also became part of her group at QuTech in Delft. The following years, I worked in both places while fulfilling the duties of a teaching assistant in Leiden. During that time, I co-authored several papers, engaged in big collaborations, software projects and presented my work at international conferences and seminars.

Bibliography

[1] R. P. Feynman. *Simulating physics with computers*. Int. J. Theor. Phys. **21**, 467 (1982).

[2] S. Lloyd. *Universal quantum simulators*. Science **273**, 1073 (1996).

[3] D. S. Abrams and S. Lloyd. *Simulation of many-body Fermi systems on a universal quantum computer*. Phys. Rev. Lett. **79**, 2586 (1997).

[4] J. Preskill. *Quantum Computing in the NISQ era and beyond*. Quantum **2**, 79 (2018).

[5] A. Y. Kitaev. *Quantum measurements and the Abelian stabilizer problem*. arXiv:quant-ph/9511026 (1995).

[6] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca. *Quantum algorithms revisited*. Proc. R. Soc. A **454**, 339 (1998).

[7] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik. *The theory of variational hybrid quantum-classical algorithms*. New J. Phys. **18**, 023023 (2016).

[8] M. Suzuki. *Fractal decomposition of exponential operators with applications to many-body theories and Monte Carlo simulations*. Phys. Lett. A **146**, 319 (1990).

[9] M. Suzuki. *General theory of fractal path integrals with applications to many-body theories and statistical physics*. J. Math. Phys. **32**, 400 (1991).

[10] G. H. Low and I. L. Chuang. *Hamiltonian simulation by qubitization*. arXiv:1610.06546 (2016).

[11] J. R. McClean, K. J. Sung, I. D. Kivlichan, Y. Cao, C. Dai, E. S. Fried, C. Gidney, B. Gimby, P. Gokhale, T. Häner, T. Hardikar, V. Havlíček, O. Higgott, C. Huang, J. Izaac, Z. Jiang, X. Liu, S. McArdle, M. Neeley, T. O'Brien, B. O'Gorman, I. Ozfidan, M. D. Radin, J. Romero, N. Rubin, N. P. D. Sawaya, K. Setia, S. Sim, D. S. Steiger, M. Steudtner, Q. Sun, W. Sun, D. Wang, F. Zhang, and R. Babbush. *OpenFermion: the electronic structure package for quantum computers*. arXiv:1710.07629 (2017).

[12] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F. J. Cabrera-Hernádez, J. Carballo-Franquis, A. Chen, C.-F. Chen, J. M. Chow, A. D. Córcoles-Gonzales, A. J. Cross, A. Cross, J. Cruz-Benito, C. Culver, S. D. L. P. González, E. D. L. Torre, D. Ding, E. Dumitrescu, I. Duran, P. Eendebak, M. Everitt, I. F. Sertage, A. Frisch, A. Fuhrer, J. Gambetta, B. G. Gago, J. Gomez-Mosquera, D. Greenberg, I. Hamamura, V. Havlicek, J. Hellmers, Ł. Herok, H. Horii, S. Hu, T. Imamichi, T. Itoko, A. Javadi-Abhari, N. Kanazawa, A. Karazeev, K. Krsulich, P. Liu, Y. Luh, Y. Maeng, M. Marques, F. J. Martín-Fernández, D. T. McClure, D. McKay, S. Meesala, A. Mezzacapo, N. Moll, D. M. Rodríguez, G. Nannicini, P. Nation, P. Ollitrault, L. J. O'Riordan, H. Paik, J. Pérez, A. Phan, M. Pistoia, V. Prutyanov, M. Reuter, J. Rice, A. R. Davila, R. H. P. Rudy, M. Ryu, N. Sathaye, C. Schnabel, E. Schoute, K. Setia, Y. Shi, A. Silva, Y. Siraichi, S. Sivarajah, J. A. Smolin, M. Soeken, H. Takahashi, I. Tavernelli, C. Taylor, P. Taylour, K. Trabing, M. Treinish, W. Turner, D. Vogt-Lee, C. Vuillot, J. A. Wildstrom, J. Wilson, E. Winston, C. Wood, S. Wood, S. Wörner, I. Y. Akhalwaya, and C. Zoufal, *Qiskit: an open-source framework for quantum computing*, 2019.

[13] G. H. Low, N. P. Bauman, C. E. Granade, B. Peng, N. Wiebe, E. J. Bylaska, D. Wecker, S. Krishnamoorthy, M. Roetteler, K. Kowalski, M. Troyer, and N. A. Baker. *Q# and NWChem: tools for scalable quantum chemistry on quantum computers*. arXiv:1904.01131 (2019).

[14] D. S. Wang, A. G. Fowler, and L. C. L. Hollenberg. *Surface code quantum computing with error rates over 1%*. Phys. Rev. A **83**, 020302 (2011).

[15] A. G. Fowler, A. M. Stephens, and P. Groszkowski. *High-threshold universal quantum computation on the surface code*. Phys. Rev. A **80**, 052312 (2009).

[16] A. Y. Kitaev. *Fault-tolerant quantum computation by anyons*. Ann. Phys. **303**, 2 (2003).

[17] H Bombin and M. A. Martin-Delgado. *Optimal resources for topological two-dimensional stabilizer codes: Comparative study*. Phys. Rev. A **76**, 012305 (2007).

[18] R. Li, L. Petit, D. P. Franke, J. P. Dehollain, J. Helsen, M. Steudtner, N. K. Thomas, Z. R. Yoscovits, K. J. Singh, S. Wehner, L. M. K. Vandersypen, J. S. Clarke, and M. Veldhorst. *A crossbar network for silicon quantum dot qubits*. Sci. Adv. **4**, eaar3960 (2018).

[19] E. P. Wigner and P. Jordan. *Über das Paulische Äquivalenzverbot*. Z. Phys. **47**, 631 (1928).

[20] T. Fujita, T. A. Baart, C. Reichl, W. Wegscheider, and L. M. K. Vandersypen. *Coherent shuttle of electron-spin states*. npj Quantum Inf. **3**, 22 (2017).

[21] R. Somma, G. Ortiz, J. E. Gubernatis, E. Knill, and R. Laflamme. *Simulating physical phenomena by quantum networks*. Phys. Rev. A **65**, 042323 (2002).

[22] R. Somma, G. Ortiz, E. Knill, and J. Gubernatis. *Quantum simulations of physics problems*. Int. J. Quantum Inf. **1**, 189 (2003).

[23] A. Aspuru-Guzik, A. D. Dutoi, P. J. Love, and M. Head-Gordon. *Simulated quantum computation of molecular energies*. Science **309**, 1704 (2005).

[24] B. P. Lanyon, J. D. Whitfield, G. G. Gillet, M. E. Goggin, M. P. Almeida, I. Kassal, J. D. Biamonte, M. Mohseni, B. J. Powell, M. Barbieri, A. Aspuru-Guzik, and A. G. White. *Towards quantum chemistry on a quantum computer*. Nat. Chem. **2**, 106 (2009).

[25] F. Verstraete and J. I. Cirac. *Mapping local Hamiltonians of fermions to local Hamiltonians of spins*. J. Stat. Mech.: Theory Exp. **2005**, P09012 (2005).

[26] S. B. Bravyi and A. Y. Kitaev. *Fermionic quantum computation*. Ann. Phys. **298**, 210 (2002).

[27] J. T. Seeley, M. J. Richard, and P. J. Love. *The Bravyi-Kitaev transformation for quantum computation of electronic structure*. J. Chem. Phys. **137**, 224109 (2012).

[28] N. C. Jones, J. D. Whitfield, P. L. McMahon, M.-H. Yung, R. Van Meter, A. Aspuru-Guzik, and Y. Yamamoto. *Faster quantum chemistry simulation on fault-tolerant quantum computers*. New J. Phys. **14**, 115023 (2012).

[29] V. Havlíček, M. Troyer, and J. D. Whitfield. *Operator locality in the quantum simulation of fermionic models*. Phys. Rev. A **95**, 032332 (2017).

[30] I. D. Kivlichan, J. McClean, N. Wiebe, C. Gidney, A. Aspuru-Guzik, G. K.-L. Chan, and R. Babbush. *Quantum simulation of electronic structure with linear depth and connectivity*. Phys. Rev. Lett. **120**, 110501 (2018).

[31] C. Tian, V. A. Vaishampayan, and N. Sloane. *Constant weight codes: a geometric approach based on dissections*. arXiv:0706.1217 (2007).

[32] S. Bravyi, G. Smith, and J. A. Smolin. *Trading classical and quantum computational resources*. Phys. Rev. X **6**, 021043 (2016).

[33] N. Moll, A. Fuhrer, P. Staar, and I. Tavernelli. *Optimizing qubit resources for quantum chemistry simulations in second quantization on a quantum computer*. J. Phys. A: Math. Theor. **49**, 295301 (2016).

[34] S. Bravyi, J. M. Gambetta, A. Mezzacapo, and K. Temme. *Tapering off qubits to simulate fermionic Hamiltonians*. arXiv:1701.08213 (2017).

[35] J. Romero, J. P. Olson, and A. Aspuru-Guzik. *Quantum autoencoders for efficient compression of quantum data*. Quantum Sci. Technol. **2**, 045001 (2017).

[36] M. Steudtner and S. Wehner. *Fermion-to-qubit mappings with varying resource requirements for quantum simulation*. New J. Phys. **20**, 063010 (2018).

[37] A. Tranter, S. Sofia, J. Seeley, M. Kaicher, J. McClean, R. Babbush, P. V. Coveney, F. Mintert, F. Wilhelm, and P. J. Love. *The Bravyi–Kitaev transformation: Properties and applications*. Int. J. Quantum Chem. **115**, 1431 (2015).

[38] J. D. Whitfield, J. Biamonte, and A. Aspuru-Guzik. *Simulation of electronic structure Hamiltonians using quantum computers*. Mol. Phys. **109**, 735 (2011).

[39] M. B. Hastings, D. Wecker, B. Bauer, and M. Troyer. *Improving quantum algorithms for quantum chemistry*. Quantum Inf. Comput. **15**, 1 (2015).

[40] A. E. Ruckenstein, P. J. Hirschfeld, and J Appel. *Mean-field theory of high-T c superconductivity: The superexchange mechanism*. Phys. Rev. B **36**, 857 (1987).

[41] P. J. J. O'Malley, R. Babbush, I. D. Kivlichan, J. Romero, J. R. McClean, R. Barends, J. Kelly, P. Roushan, A. Tranter, N. Ding, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, A. G. Fowler, E. Jeffrey, A. Megrant, J. Y. Mutus, C. Neill, C. Quintana, D. Sank, A. Vainsencher, J. Wenner, T. C. White, P. V. Coveney, P. J. Love, H. Neven, A. Aspuru-Guzik, and J. M. Martinis. *Scalable quantum simulation of molecular energies*. Phys. Rev. X **6**, 031007 (2016).

[42] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. *Elementary gates for quantum computation*. Phys. Rev. A **52**, 3457 (1995).

[43] M. Steudtner and S. Wehner. *Quantum codes for quantum simulation of fermions on a square lattice of qubits*. Phys. Rev. A **99**, 022308 (2019).

[44] J. Du, N. Xu, X. Peng, P. Wang, S. Wu, and D. Lu. *NMR implementation of a molecular hydrogen quantum simulation with adiabatic state preparation*. Phys. Rev. Lett. **104**, 030502 (2010).

[45] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien. *A variational eigenvalue solver on a photonic quantum processor*. Nat. Commun. **5**, 4213 (2014).

[46] R. Barends, L. Lamata, J. Kelly, L. García-Álvarez, A. G. Fowler, A. Megrant, E. Jeffrey, T. C. White, D. Sank, J. Y. Mutus, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, I. C. Hoi, C. Neill, P. J. J. O'Malley, C. Quintana, P. Roushan, A. Vainsencher, J. Wenner, E. Solano, and J. M. Martinis. *Digital quantum simulation of*

fermionic models with a superconducting circuit. Nat. Commun. **6**, 7654 (2015).

[47] Y. Wang, F. Dolde, J. Biamonte, R. Babbush, V. Bergholm, S. Yang, I. Jakobi, P. Neumann, A. Aspuru-Guzik, J. D. Whitfield, and J. Wrachtrup. *Quantum simulation of helium hydride cation in a solid-state spin register*. ACS nano **9**, 7769 (2015).

[48] C. Hempel, C. Maier, J. Romero, J. McClean, T. Monz, H. Shen, P. Jurcevic, B. Lanyon, P. Love, R. Babbush, A. Aspuru-Guzik, R. Blatt, and C. Roos. *Quantum chemistry calculations on a trapped-ion quantum simulator*. Phys. Rev. X **8**, 031022 (2018).

[49] D. Wecker, B. Bauer, B. K. Clark, M. B. Hastings, and M. Troyer. *Gate-count estimates for performing quantum chemistry on small quantum computers*. Phys. Rev. A **90**, 022305 (2014).

[50] T Holstein and H. Primakoff. *Field dependence of the intrinsic domain magnetization of a ferromagnet*. Phys. Rev. **58**, 1098 (1940).

[51] E. Fradkin. *Jordan-Wigner transformation for quantum-spin systems in two dimensions and fractional statistics*. Phys. Rev. Lett. **63**, 322 (1989).

[52] Y. Wang. *Ground state of the two-dimensional antiferromagnetic Heisenberg model studied using an extended Wigner-Jordon transformation*. Phys. Rev. B **43**, 3786 (1991).

[53] R. Ball. *Fermions without fermion fields*. Phys. Rev. Lett. **95**, 176407 (2005).

[54] Y.-A. Chen, A. Kapustin, and Đ. Radičević. *Exact bosonization in two spatial dimensions and a new class of lattice gauge theories*. Ann. Phys. **393**, 234 (2018).

[55] J. D. Whitfield, V. Havlíček, and M. Troyer. *Local spin operators for fermion simulations*. Phys. Rev. A **94**, 030301 (2016).

[56] E. Zohar and J. I. Cirac. *Eliminating fermionic matter fields in lattice gauge theories*. Phys. Rev. B **98**, 075119 (2018).

[57] K. Setia and J. D. Whitfield. *Bravyi-Kitaev Superfast simulation of fermions on a quantum computer*. arXiv:1712.00446 (2017).

[58] R. Beals, S. Brierley, O. Gray, A. W. Harrow, S. Kutin, N. Linden, D. Shepherd, and M. Stather. *Efficient distributed quantum computing*. Proc. R. Soc. A **469**, 20120686 (2013).

[59] R. Babbush, N. Wiebe, J. McClean, J. McClain, H. Neven, and G. K.-L. Chan. *Low-depth quantum simulation of materials*. Phys. Rev. X **8**, 011044 (2018).

[60] Y. Subaşı and C. Jarzynski. *Nonperturbative embedding for highly nonlocal Hamiltonians*. Phys. Rev. A **94**, 012342 (2016).

[61] D. Poulin, A. Kitaev, D. S. Steiger, M. B. Hastings, and M. Troyer. *Quantum algorithm for spectral measurement with a lower gate count*. Phys. Rev. Lett. **121**, 010501 (2018).

[62] R. Babbush, C. Gidney, D. W. Berry, N. Wiebe, J. McClean, A. Paler, A. Fowler, and H. Neven. *Encoding electronic spectra in quantum circuits with linear T complexity*. Phys. Rev. X **8**, 041015 (2018).

[63] R. J. Bartlett, S. A. Kucharski, and J. Noga. *Alternative coupled-cluster ansätze ii. the unitary coupled-cluster method*. Chem. Phys. Lett. **155**, 133 (1989).

[64] F. Motzoi, M. Kaicher, and F. Wilhelm. *Linear and logarithmic time compositions of quantum many-body operators*. Phys. Rev. Lett. **119**, 160503 (2017).

[65] D. Poulin, M. B. Hastings, D. Wecker, N. Wiebe, A. C. Doberty, and M. Troyer. *The trotter step size required for accurate quantum simulation of quantum chemistry*. Quantum Information & Computation **15**, 361 (2015).

[66] A. M. Childs, A. Ostrander, and Y. Su. *Faster quantum simulation by randomization*. arXiv:1805.08385 (2018).

[67] E. Campbell. *A random compiler for fast hamiltonian simulation*. arXiv:1811.08017 (2018).

[68] J. Haah, M. B. Hastings, R. Kothari, and G. H. Low. *Quantum algorithm for simulating real time evolution of lattice Hamiltonians*. arXiv:1801.03922 (2018).

[69] D. Wecker, M. B. Hastings, N. Wiebe, B. K. Clark, C. Nayak, and M. Troyer. *Solving strongly correlated electron models on a quantum computer*. Phys. Rev. A **92**, 062318 (2015).

[70] S. McArdle, X. Yuan, and S. Benjamin. *Error-mitigated digital quantum simulation*. Phys. Rev. Lett. **122**, 180501 (2019).

[71] X Bonet-Monroig, R Sagastizabal, M Singh, and T. O'Brien. *Low-cost error mitigation by symmetry verification*. Phys. Rev. A **98**, 062339 (2018).

[72] R Versluis, S Poletto, N Khammassi, B Tarasinski, N Haider, D. Michalak, A Bruno, K Bertels, and L DiCarlo. *Scalable quantum circuit and control for a superconducting surface code*. Phys. Rev. Appl. **8**, 034021 (2017).

[73] G. Zhu, Y. Subaşı, J. D. Whitfield, and M. Hafezi. *Hardware-efficient fermionic simulation with a cavity–QED system*. npj Quantum Inf. **4**, 16 (2018).

[74] Z. Jiang, J. McClean, R. Babbush, and H. Neven. *Majorana loop stabilizer codes for error correction of fermionic quantum simulations*. arXiv:1812.08190 (2018).

[75] K. Setia, S. Bravyi, A. Mezzacapo, and J. D. Whitfield. *Superfast encodings for fermionic quantum simulation*. arXiv:1810.05274 (2018).

[76] J. Colless. *Control and readout of scaled-up quantum dot systems*. PhD Thesis, University of Sydney (2014).

[77] L. M. K. Vandersypen, H. Bluhm, J. S. Clarke, A. S. Dzurak, R. Ishihara, A. Morello, D. J. Reilly, L. R. Schreiber, and M. Veldhorst. *Interfacing spin qubits in quantum dots and donors—hot, dense, and coherent*. npj Quantum Inf. **3**, 34 (2017).

[78] C. D. Hill, E. Peretz, S. J. Hile, M. G. House, M. Fuechsle, S. Rogge, M. Y. Simmons, and L. C. Hollenberg. *A surface code quantum computer in silicon*. Sci. Adv. **1**, e1500707 (2015).

[79] M. Veldhorst, H. G. J. Eenink, C. H. Yang, and A. S. Dzurak. *Silicon CMOS architecture for a spin-based quantum computer*. Nat. Commun. **8**, 1766 (2017).

[80] D. Gottesman. *Theory of fault-tolerant quantum computation*. Phys. Rev. A **57**, 127 (1998).

[81] D. A. Lidar, T. A. Brun, and T. Brun, eds. *Quantum error correction*. (Cambridge University Press, 2009).

[82] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill. *Topological quantum memory*. J. Math. Phys. **43**, 4452 (2002).

[83] H. Bombin and M. A. Martin-Delgado. *Topological quantum distillation*. Phys. Rev. Lett. **97**, 180501 (2006).

[84] T. Heijmen. "Soft errors from space to ground: historical overview, empirical evidence, and future trends". In *Soft errors in modern electronic systems* (Springer US, 2010), pp. 1–25.

[85] R. Hanson, L. P. Kouwenhoven, J. R. Petta, S. Tarucha, and L. M. K. Vandersypen. *Spins in few-electron quantum dots*. Rev. Mod. Phys. **79**, 1217 (2007).

[86] J. M. Taylor, H.-A. Engel, W. Dür, A. Yacoby, C. M. Marcus, P. Zoller, and M. D. Lukin. *Fault-tolerant architecture for quantum computation using electrically controlled semiconductor spins*. Nat. Phys. **1**, 177 (2005).

[87] M. Veldhorst, J. C. C. Hwang, C. H. Yang, A. W. Leenstra, B. de Ronde, J. P. Dehollain, J. T. Muhonen, F. E. Hudson, K. M. Itoh, A. Morello, and A. S. Dzurak. *An addressable quantum dot qubit with fault-tolerant control-fidelity*. Nat. Nanotechnol. **9**, 981 (2014).

[88] M. A. Nielsen and I. Chuang. *Quantum computation and quantum information*. AAPT (2002).

[89] J. R. Petta. *Coherent manipulation of coupled electron spins in semiconductor quantum dots*. Science **309**, 2180 (2005).

[90] T. Meunier, V. E. Calado, and L. M. K. Vandersypen. *Efficient controlled-phase gate for single-spin qubits in quantum dots*. Phys. Rev. B **83**, 121403 (2011).

[91] T. F. Watson, S. G. J. Philips, E. Kawakami, D. R. Ward, P. Scarlino, M. Veldhorst, D. E. Savage, M. G. Lagally, M. Friesen, S. N. Coppersmith, M. A. Eriksson, and L. M. K. Vandersypen. *A programmable two-qubit quantum processor in silicon*. Nature **555**, 633 (2018).

[92] M. Veldhorst, C. H. Yang, J. C. C. Hwang, W. Huang, J. P. Dehollain, J. T. Muhonen, S. Simmons, A. Laucht, F. E. Hudson, K. M. Itoh, A. Morello, and A. S. Dzurak. *A two-qubit logic gate in silicon*. Nature **526**, 410 (2015).

[93] N. Schuch and J. Siewert. *Natural two-qubit gate for quantum computation using the XY interaction*. Phys. Rev. A **67**, 032301 (2003).

[94] J. Helsen, M. Steudtner, M. Veldhorst, and S. Wehner. *Quantum error correction in crossbar architectures*. Quantum Sci. Technol. **3**, 035005 (2018).

[95] J. Helsen. *Quantum computing in the real world, Diagnosing and correcting errors in practical quantum devices*. PhD thesis, TU Delft (2019).

[96] B. M. Terhal. *Quantum error correction for quantum memories*. Rev. Mod. Phys. **87**, 307 (2015).

[97] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland. *Surface codes: Towards practical large-scale quantum computation*. Phys. Rev. A **86**, 032324 (2012).

[98] Y. Tomita and K. M. Svore. *Low-distance surface codes under realistic quantum noise*. Phys. Rev. A **90**, 062320 (2014).

[99] A. M. Tyryshkin, S. Tojo, J. J. L. Morton, H. Riemann, N. V. Abrosimov, P. Becker, H.-J. Pohl, T. Schenkel, M. L. W. Thewalt, K. M. Itoh, and S. A. Lyon. *Electron spin coherence exceeding seconds in high-purity silicon*. Nat. Mater. **11**, 143 (2011).

[100] A. J. Landahl, J. T. Anderson, and P. R. Rice. *Fault-tolerant quantum computing with color codes*. arXiv:1108.5738 (2011).

[101] J. R. Wootton, A. Peter, J. R. Winkler, and D. Loss. *Proposal for a minimal surface code experiment*. Phys. Rev. A **96**, 032338 (2017).

[102] D. K. Tuckett, S. D. Bartlett, and S. T. Flammia. *Ultrahigh error threshold for surface codes with biased noise*. Phys. Rev. Lett. **120**, 050505 (2018).

[103] C. Horsman, A. G. Fowler, S. Devitt, and R. V. Meter. *Surface code quantum computing by lattice surgery*. New J. Phys. **14**, 123011 (2012).