Computational Methods for Simulating Quantum Computers *

H. De Raedt† and K. Michielsen‡
Department of Applied Physics, Materials Science Centre,
University of Groningen, Nijenborgh 4, NL-9747 AG Groningen, The Netherlands§
(Dated: January 9, 2005)

This review gives a survey of numerical algorithms and software to simulate quantum computers. It covers the basic concepts of quantum computation and quantum algorithms and includes a few examples that illustrate the use of simulation software for ideal and physical models of quantum computers.

Keywords: Quantum computation, computer simulation, time-integration algorithms

Contents

---

* To appear in: Handbook of Theoretical and Computational Nanotechnology (American Scientific Publishers).
†Electronic address: deraedt@phys.rug.nl
‡Electronic address: kristel@phys.rug.nl
§URL: http://www.compphys.org

I. INTRODUCTION

The basic ideas of quantum computation were formulated more than 20 years ago [1, 2]. In the 1990's several algorithms have been discovered [3–8] that run much faster on a quantum computer than on a conventional computer, promising the solution of some problems that are considered to be intractable for conventional computers. These discoveries have fueled many new developments, both theoretically and experimentally. Quantum information theory is a new interdisciplinary field of research, building on concepts from theoretical computer science and theoretical physics [9]. On the experimental front, considerable progress has been made to engineer and control quantum systems that may be used for quantum information processing [10–32]. The feasibility of executing short quantum algorithms on quantum systems with a few (< 8 bits) has been demonstrated [14–17, 29–31]. The technological challenges to build a quantum computer that can perform calculations that would exhaust the resources of a conventional computer seem tremendous, and there is no indication that such a machine will become available in the next few years.

Any physically realizable quantum computer is a complicated many-body system that interacts with its environment. In quantum statistical mechanics and quantum chemistry, it is well known that simulating an interacting quantum many-body system becomes exponentially more difficult as the size of the system grows. Actually, it is this observation that lead Feynman to the question what kind of computer would be needed to overcome this exponential increase [2]. Formally, the answer to this question is "a quantum computer." It seems that only a quantum computer can efficiently simulate itself [2, 33].

Computer simulation has long been accepted as the third methodology in many branches of science and engineering [34]. Conventional computers can be used to simulate quantum computers that are relatively small (such as 24 qubits) but are significantly larger than the experimental machines that have been built. Therefore, it is striking that theoretical ideas about quantum computation are seldom confronted with numerical experiments that can be carried out on present-day (super) computers. Conventional computers can simulate the abstract model of an ideal quantum computer on which most theoretical work is based, and, most important, they can also simulate the physical behavior of quantum computer hardware [35].

There are a number of excellent reviews [31, 36–42] and books [9, 43–45] that cover the theoretical and/or experimental aspects of quantum computations, but there is no review that discusses methods to simulate ideal and physical models of quantum computers. The purpose of this review is to fill this void. It contains enough information and detail to allow advanced students to develop their own quantum computer simulator. It also contains an up-to-date account of existing simulation software. Most of the simulation methods covered in this review have a much broader scope than the application to quantum computation might suggest. As a matter of fact, much of the impetus to develop these methods stems from problems encountered in the study of quantum dynamics of nanomagnets.

This chapter is organized as follows. Section II gives a brief account of the basics of quantum computation. Section III discusses the implemention of simulation models of quantum computers on a conventional computer and also gives a survey of the numerical algorithms that are used to simulate these models. A short review of quantum algorithms is given in Section IV. Sections V and VI deal with the simulation of ideal and realistic models of quantum computers, respectively. In Section VII we give an overview of existing quantum computer simulator software and also provide examples of how a simulator is used to perform quantum computation. A brief summary and outlook are provided in Section VIII.

II. THE IDEAL QUANTUM COMPUTER

In contrast to a digital computer, in which the state of an elementary storage unit is specified by the logical values 0 and 1 (= one bit), the state of an elementary storage unit of a quantum computer, the quantum bit or qubit, is described by a two-dimensional vector of Euclidean length one. Denoting two orthogonal basis vectors of the two-dimensional vector space by $|0\rangle$ and $|1\rangle$, the state $|\Phi\rangle$ of the **qubit** can be written as a linear superposition of the basis states $|0\rangle$ and $|1\rangle$:

$$|\Phi\rangle = a_0|0\rangle + a_1|1\rangle, \tag{1}$$

where a_0 and a_1 are complex numbers such that $|a_0|^2 + |a_1|^2 = 1$. The appearance of complex numbers suggests that one qubit can contain an infinite amount of information. However, the principles of quantum mechanics, do not allow retrieving all this information [46–48]. The result of inquiring about the state of the qubit, that is a measurement, yields either the result 0 or 1. The frequency of obtaining 0 (1) can be estimated by repeated measurement of the same state of the qubits and is given by $|a_0|^2$ ($|a_1|^2$).

Just as a conventional computer with only one bit is fairly useless, a quantum computer should also contain more than one qubit. Disregarding the enormous technological hurdles that have to be taken to put more than, say 7, qubits together, on present-day personal computers we can readily simulate quantum computers with $L = 20$ qubits. The internal state of a quantum computer with L qubits is represented by a unit vector in a $D = 2^L$ dimensional space (see Section II B).

The ideal quantum computer differs from a conventional (probabilistic) computer in the sense that the state of the qubits changes according to the rules of quantum mechanics, that is through rotations of the vectors representing the state of the qubits [9]. From elementary linear algebra, we know that a rotation of a vector corresponds to the multiplication of the vector by a unitary matrix. Thus, the internal state of the quantum computer evolves in time according to a sequence of unitary transformations. Any such sequence is called a **quantum algorithm**. Of course, not every sequence corresponds to a meaningful computation. If a quantum algorithm cannot exploit the fact that the intermediate state of the quantum computer is described by a linear superposition of basis vectors, it will not be faster than its classical counterpart.

The unitary time evolution of the internal state of the quantum computer is interrupted at the point where we inquire about the value of the qubits, that is as soon as we perform a measurement on the qubits. If we perform the readout operation on a qubit, we get a definite answer, either 0 or 1, and the information encoded in the superposition is lost. The process of measurement cannot be described by a unitary transformation [48]. Therefore, we do not consider it to be part of the quantum algorithm. However, to estimate the efficiency of a quantum computation, both the operation count of the quantum algorithm and the cost of measuring the outcome of the calculation have to be taken into consideration [49].

In the quantum computation literature the convention, is to count each application of a unitary transformation as one operation on a quantum computer [9]. As the unitary transformation may change all amplitudes simultaneously, a quantum computer is a massively parallel machine. To simulate these unitary operations on a conventional computer, we actually perform matrix-vector multiplications that requires $\mathcal{O}(2^{2L})$ (in the worst case) arithmetic operations. This may cause some confusion at some points, so the reader should try to keep this in mind.

To summarize: At any point in time, the internal state of a quantum computer with L qubits is described by a unit vector in a $D = 2^L$ dimensional space. A quantum algorithm that executes on the quantum computer changes the internal state by rotating the vector. Therefore, each step in the quantum algorithm corresponds to the multiplication of the vector by a unitary matrix. Readout of the result of a quantum algorithm is destructive in the sense that it destroys the information contained in the superposition.

In the following subsections, we discuss the basic ingredients of quantum computation in more detail.

A. Spin-1/2 Algebra

In the simplest form, a qubit is a two-state quantum system, of which there are many examples in the quantum world [46–48]. Measurements of a component of the spin (= internal angular momentum) of particles such as electrons, protons, and neutrons along any direction yield either $\hbar/2$ or $-\hbar/2$. The convention is to call the state of the spin that corresponds to the outcome $\hbar/2$ "spin up" ($|\uparrow\rangle$) and the other state "spin down" ($|\downarrow\rangle$). The fact that the internal angular momentum takes only two values implies that it corresponds to a total angular momentum quantum number of 1/2, hence the name spin 1/2 ($S = 1/2$) particle.

In principle, any physical object that carries a $S = 1/2$ degree of freedom can be used as a physical realization of a qubit. In general, the wavefunction that describes the spin of these objects can be written as a linear combination of the spin-up and spin-down states [46–48]:

$$|\Phi\rangle = a_0|\uparrow\rangle + a_1|\downarrow\rangle, \tag{2}$$

where a_0 and a_1 are complex numbers. It is convenient to normalize the length of the vector $|\Phi\rangle$ to one. Then $|a_0|^2 + |a_1|^2 = 1$.

The three components of the spin-1/2 operator $\mathbf{S}$ acting on the Hilbert space spanned by the states $|\uparrow\rangle$ and $|\downarrow\rangle$ are defined by [46–48]

$$S^x = \frac{1}{2}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad S^y = \frac{1}{2}\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad S^z = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \tag{3}$$

in units such that $\hbar = 1$. According to quantum theory, a measurement of a physical quantity $A = A^\dagger$ of a quantum system in a state $|\Phi\rangle$ yields a real number, the **expectation value** [46–48]

$$\langle A\rangle = \langle\Phi|A|\Phi\rangle/\langle\Phi|\Phi\rangle. \tag{4}$$

From Eqs. (2) and (3) it follows that

$$\langle S^x\rangle = \frac{a_0a_1^* + a_0^*a_1}{2} \quad , \langle S^y\rangle = \frac{ia_0a_1^* - ia_0^*a_1}{2} \quad , \langle S^z\rangle = \frac{a_0a_0^* - a_1^*a_1}{2} \quad , \tag{5}$$

and we conclude that the representations Eqs. (2) and (3), have been chosen such that $|\uparrow\rangle$ and $|\downarrow\rangle$ are eigenstates of S^z with eigenvalues $+1/2$ and $-1/2$, respectively. This is the convention adopted in the physics literature.

In the context of quantum computation, it is convenient to define [9]

$$|0\rangle = |\uparrow\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad , \quad |1\rangle = |\downarrow\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{6}$$

Then we have

$$|\Phi\rangle = a_0|0\rangle + a_1|1\rangle, \tag{7}$$

and the expectation values of the three components of the qubits are defined as

$$\langle Q^x\rangle = \frac{1 - a_0a_1^* - a_0^*a_1}{2} \quad , \quad \langle Q^y\rangle = \frac{1 - ia_0a_1^* + ia_0^*a_1}{2} \quad , \quad \langle Q^z\rangle = \frac{1 - a_0a_0^* + a_1^*a_1}{2} \quad , \tag{8}$$

such that $0 \leq \langle Q^\alpha\rangle \leq 1$ for $\alpha = x, y, z$. In the following, when we say that the state of a qubit is 0(1), we actually mean that $\langle Q^z\rangle = 0(1)$ and $\langle Q^x\rangle = \langle Q^y\rangle = 1/2$.

A change of the state of a qubit corresponds to a rotation of the spin. In general, a rotation of the spin by an angle ϕ about an axis β can be written as ($\hbar = 1$)

$$S^\alpha(\phi, \beta) = e^{i\phi S^\beta} S^\alpha e^{-i\phi S^\beta} = S^\alpha \cos\phi + \epsilon_{\alpha\beta\gamma} S^\gamma \sin\phi, \tag{9}$$

where use has been made of the commutation rules of the components of the angular momentum operator $\mathbf{S}$ [46–48]:

$$[S^\alpha, S^\beta] = i\epsilon_{\alpha\beta\gamma} S^\gamma, \tag{10}$$

where $\epsilon_{\alpha\beta\gamma}$ is the totally asymmetric unit tensor ($\epsilon_{xyz} = \epsilon_{yzx} = \epsilon_{zxy} = 1$, $\epsilon_{\alpha\beta\gamma} = -\epsilon_{\beta\alpha\gamma} = -\epsilon_{\gamma\beta\alpha} = -\epsilon_{\alpha\gamma\beta}$, $\epsilon_{\alpha\alpha\gamma} = 0$) and the summation convention is used. In addition to Eq. (10) the $S = 1/2$ operators (3) also satisfy the relations [46–48]

$$S^xS^x = S^yS^y = S^zS^z = \frac{1}{4}, \tag{11}$$

$$S^xS^y = \frac{i}{2}S^z, \quad S^xS^z = -\frac{i}{2}S^y, \quad S^yS^z = \frac{i}{2}S^x, \quad S^yS^x = -\frac{i}{2}S^z, \quad S^zS^x = \frac{i}{2}S^y, \quad S^zS^y = -\frac{i}{2}S^x, \tag{12}$$

which are often very useful to simplify products of $S = 1/2$ matrices. From Eq. (12) it follows that $2S^x$, $2S^y$, and $2S^z$ are unitary operators. Hence each of them represents a genuine quantum computer operation.

Rotations of the spin by $\pi/2$ about the x-axis and y-axis are often used as elementary quantum computer operations. In matrix notation, they are given by

$$X \equiv e^{i\pi S^x/2} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix}, \quad Y \equiv e^{i\pi S^y/2} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}, \tag{13}$$

respectively. X and Y represent operations on single qubits. The matrix expressions for the inverse of the rotations X and Y, denoted by $\overline{X}$ and $\overline{Y}$, respectively, are obtained by taking the Hermitian conjugates of the matrices in (13).

With our convention, $\langle 0|\overline{Y}S^xY|0\rangle = -1/2$ so that a positive angle corresponds to a rotation in the clockwise direction. In general, a rotation of the state about a vector $\mathbf{v}$ corresponds to the matrix

$$e^{i\mathbf{v}\cdot\mathbf{S}} = \mathbb{1}\cos\frac{v}{2} + \frac{2i\mathbf{v}\cdot\mathbf{S}}{v}\sin\frac{v}{2}, \tag{14}$$

where $\mathbb{1}$ denotes the unit matrix and $v = \sqrt{v_x^2 + v_y^2 + v_z^2}$ is the length of the vector $\mathbf{v}$.

B. State Representation

A quantum computer must have more than one qubit. Let us denote by L the number of $S = 1/2$ objects of the corresponding physical system. In general, the quantum state of a system of L spins is defined by the linear superposition of the direct-product states of the L single-spin states:

$$|\Phi\rangle = \quad a(\uparrow\uparrow\ldots\uparrow)|\uparrow\uparrow\ldots\uparrow\rangle + a(\downarrow\uparrow\ldots\uparrow)|\downarrow\uparrow\ldots\uparrow\rangle + \ldots + a(\uparrow\downarrow\ldots\downarrow)|\uparrow\downarrow\ldots\downarrow\rangle + a(\downarrow\downarrow\ldots\downarrow)|\downarrow\downarrow\ldots\downarrow\rangle. \tag{15}$$

As before, the coefficients (amplitudes) $a(\uparrow\uparrow\ldots\uparrow), \ldots, a(\downarrow\downarrow\ldots\downarrow)$ are complex numbers, and it is convenient to normalize the vector $|\phi\rangle$ by the rescaling

$$\sum_{\sigma_1\ldots\sigma_L=\uparrow,\downarrow} |a(\sigma_1\ldots\sigma_L)|^2 = 1, \tag{16}$$

such that $\langle\Phi|\Phi\rangle = 1$.

In physics, it is customary to let the first label correspond to the first spin, the second to the second spin, and so on. Thus, the second term in Eq. (15) is the contribution of the state with spin 1 down and all other spins up. In computer science, it is more natural (and logically equivalent) to think of qubit (spin) 1 as the least significant bit of the integer index that runs from zero (all spins up) to $2^L - 1$ (all spins down). This is also the convention that is adopted in the literature on quantum computation: spin up (down) corresponds to a qubit in state 0 (1). Thus, in quantum-computation notation Eq. (15) reads

$$\begin{aligned} |\Phi\rangle &= a(0\ldots00)|0\ldots00\rangle + a(0\ldots01)|0\ldots01\rangle + \ldots + a(1\ldots10)|1\ldots10\rangle + a(1\ldots11)|1\ldots11\rangle, \\ &= a_0|0\rangle + a_1|1\rangle + \ldots + a_{2^L-2}|2^L-2\rangle + a_{2^L-1}|2^L-1\rangle. \end{aligned} \tag{17}$$

For later reference, it is useful to write down explicitly some examples of translating from physics to quantum computer notation and vice versa:

$$
\begin{aligned}
a(\uparrow\uparrow\uparrow\ldots\uparrow)|\uparrow\uparrow\uparrow\ldots\uparrow\rangle &= a(0\ldots000)|0\ldots000\rangle = a_0|0\rangle,\\
a(\downarrow\uparrow\uparrow\ldots\uparrow)|\downarrow\uparrow\uparrow\ldots\uparrow\rangle &= a(0\ldots001)|0\ldots001\rangle = a_1|1\rangle,\\
a(\uparrow\downarrow\uparrow\ldots\uparrow)|\uparrow\downarrow\uparrow\ldots\uparrow\rangle &= a(0\ldots010)|0\ldots010\rangle = a_2|2\rangle,\\
a(\uparrow\downarrow\downarrow\ldots\downarrow)|\uparrow\downarrow\downarrow\ldots\downarrow\rangle &= a(1\ldots110)|1\ldots110\rangle = a_{2^L-2}|2^L-2\rangle,\\
a(\downarrow\downarrow\downarrow\ldots\downarrow)|\downarrow\downarrow\downarrow\ldots\downarrow\rangle &= a(1\ldots111)|1\ldots111\rangle = a_{2^L-1}|2^L-1\rangle. \qquad (18)
\end{aligned}
$$

Obviously, there is a one-to-one correspondence between the last line of Eq. (17) and the way the amplitudes a_i are stored in the memory of a conventional computer. From representation (17), it is easy to estimate the amount of computer memory that is needed to simulate a quantum spin system of L spins on a conventional digital computer. The dimension D of the Hilbert space (that is the number of amplitudes a_i) spanned by the L spin-1/2 states is $D = 2^L$. For applications that require highly optimized code, it is often more efficient to store the real and imaginary part of the amplitudes a_i in separate arrays. Furthermore, even for simple looking problems, it is advisable to use 13 - 15 digit floating-point arithmetic (corresponding to 8 bytes for a real number). Thus, to represent a state of the quantum system of L qubits in a conventional, digital computer, we need a least 2^{L+4} bytes. For example, for $L = 20$ ($L = 24$) we need at least 16 (256) Mb of memory to store a single arbitrary state $|\Phi\rangle$. Not surprisingly, the amount of memory that is required to simulate a quantum system with L spins increases exponentially with the number of spins L.

Operations on the jth spin are denoted by simply attaching the label j to the three components of the spin operator. For example, the operation that flips the second spin (or qubit) is given by

$$
\begin{aligned}
|\Phi'\rangle &= S_2^x|\Phi\rangle\\
&= a(\uparrow\uparrow\uparrow\ldots\uparrow)|\uparrow\downarrow\uparrow\ldots\uparrow\rangle + a(\downarrow\uparrow\uparrow\ldots\uparrow)|\downarrow\downarrow\uparrow\ldots\uparrow\rangle + \ldots\\
&\quad + a(\uparrow\downarrow\downarrow\ldots\downarrow)|\uparrow\uparrow\downarrow\ldots\downarrow\rangle + a(\downarrow\downarrow\downarrow\ldots\downarrow)|\downarrow\uparrow\downarrow\ldots\downarrow\rangle\\
&= a(0\ldots000)|0\ldots010\rangle + a(0\ldots001)|0\ldots011\rangle + \ldots\\
&\quad + a(1\ldots110)|1\ldots100\rangle + a(1\ldots111)|1\ldots101\rangle. \qquad (19)
\end{aligned}
$$

It is important to note that although the spin operator S_j^x only flips the jth spin, to compute $|\Phi'\rangle$ on a conventional computer we have to update *all* the 2^L amplitudes. In general, on a conventional computer *each* operation on the wavefunction involves at least 2^L arithmetic operations (here and in the sequel, if we count operations on a conventional computer, we make no distiction between operations such as add, multiply, or get from and put to memory). The fact that both the amount of memory and arithmetic operation count increase exponentially with the number of qubits is the major bottleneck for simulating quantum systems (including quantum computers) on a conventional computer. However, at the time of writing, the number of qubits that can be simulated far exceeds the number of qubits of experimentally realizable systems.

C. Universal Computer

It is easy to see that any (Boolean) logic circuit of a conventional computer can be built by an appropriate combination of NAND-gates [50]. Although in practice it is often convenient to think in terms of logic circuits that can perform more complex tasks than a NAND operation, conceptually it is important to know that NAND-gates are all that is needed to build a universal computing device. Also, a quantum computer can be constructed from a set of basic gates (= unitary transformations), but instead of one gate we need several [9, 36, 38, 51–53]. The minimal set of gates is not unique, but, for ideal quantum computers, it is mainly a matter of taste to choose a particular set. However, in the real world, the kind of operations we can perform on a specific physical system is limited, and this will bias the choice of the set of basic gates. Nevertheless, as long as the chosen set of basic gates allows us to synthesize *any* unitary tranformation on the state $|\Phi\rangle$ of the quantum computer, this set can be used for universal quantum computation [9, 54].

In analogy with Boolean logic circuits, simplicity is an important guideline to determine the set of elementary unitary gates of a quantum computer. Some basic results from linear algebra are very helpful in this respect. It is well known that any vector of N elements can be transformed to the vector $(1, 0, \ldots, 0)$ by at most $N - 1$ plane rotations [55, 56]. Each plane rotation is represented by a 2×2 unitary matrix that operates on the elements $(1, j)$ for $j = 2, \ldots, N$. Application of this procedure to the N, mutually orthogonal, column vectors of a $N \times N$ unitary matrix U immediately leads to the conclusion that U can be written as a product of at most $N(N - 1)/2$ plane rotations.

Thus, we can synthesize *any* $N \times N$ unitary matrix by multiplying unitary matrices each of which involves only two of the N basis states [57].

As the order in which we apply the plane rotations is arbitrary, the decomposition is not unique. Furthermore, on a quantum computer $N = 2^L$ so that for a given quantum algorithm (or equivalently, a given unitary matrix U) we should be able to find a decomposition that takes much less than $2^{L-1}(2^L - 1)$ plane rotations. Otherwise the implementation of the quantum algorithm is not efficient, and there is no point of even contemplating the use of a quantum computer to solve the problem. Finding the optimal decomposition of a $N \times N$ unitary matrix U in terms of plane rotations is a difficult computational problem, in particular because it is known that for some U there is no other alternative than to decompose it into $N(N-1)/2$ plane rotations [9]. It has been shown that any of the $2^{L-1}(2^L - 1)$ plane rotations can be constructed by a combination of single qubit gates and the so-called CNOT gate that operates on two qubits [9, 53]. Therefore these single qubit operations and the CNOT gate constitute a set of gates that can be used to construct a universal quantum computer.

Another basic result of linear algebra is that any nonsingular matrix can be written as the matrix exponential of a Hermitian matrix [58]. As a unitary matrix is nonsingular, we can write $U = e^{-itH}$, where we already anticipated that, for the case at hand, the Hermitian matrix H corresponds to the Hamiltonian that models the qubits. The next question is then to ask for the class of model Hamiltonians that can be used for universal computation. Again, simplicity is an important criterion to determine useful models, but, as the Hamiltonian represents a physical system, there may be some additional constraints on the form of the interactions among qubits and the like. We discuss these aspects in more detail in Section VI, where we consider models of physically realizable quantum computers.

It is a remarkable fact that the simplest quantum many-body system, the Ising spin model, can be used for universal quantum computation [51]. The Ising model

$$H(t) = -\sum_{i,j=1}^{L} J_{i,j}^{z}(t) S_i^z S_j^z - \sum_{i=1}^{L} \sum_{\alpha=x,y,z} h_i^{\alpha}(t) S_i^{\alpha}, \tag{20}$$

is often used as a physical model of an ideal universal quantum computer [9, 51, 59]. The first sum in Eq. (20) runs over all pairs $P \leq (L-1)L/2$ of spins that interact with each other. For instance, if the spins interact only with their nearest neighbors, we have $P = L/2$ and $J_{i,j}^{z}(t) = 0$ except when i and j refer to spins that are neighbors. In the case of dipolar interaction, $P = (L-1)L/2$. Note that Eq. (20) implicitly assumes that we have complete control over all interaction parameters $J_{i,j}^{z}(t)$ and external fields $h_i^{\alpha}(t)$. In Sections II E and II F we explain how a universal set of gates, the single qubit and the CNOT gates, can be implemented on the quantum computer defined by the Ising model (20).

The more general form of the Hamiltonian of a physical model of a quantum computer reads

$$H(t) = -\sum_{i,j=1}^{L} \sum_{\alpha=x,y,z} J_{i,j}^{\alpha}(t) S_i^{\alpha} S_j^{\alpha} - \sum_{i=1}^{L} \sum_{\alpha=x,y,z} h_i^{\alpha}(t) S_i^{\alpha}. \tag{21}$$

The exchange parameters $J_{i,j}^{\alpha}(t)$ determine the strength of the interactions between the α-components of spins i and j.

Hamiltonian (21) is sufficiently generic to represent most models for candidates of physical realizations of quantum computer hardware. The spin-spin term in Eq. (21) is sufficiently general to describe the most common types of interactions such as Ising, anisotropic Heisenberg, and dipolar coupling between the spins. Furthermore, if we also use spin-1/2 degrees of freedom to represent the environment then, on this level of description, the interaction between the quantum computer and its environment is included in model (21), too. In other words, the Hamiltonian (21) is sufficiently generic to cover most cases of current interest.

In the context of quantum computation and experiments in general, the quantum dynamics of model (21) is controlled and/or probed by static and/or time-dependent external fields $h_i^{\alpha}(t)$. Depending on the physical system, also the exchange parameters $J_{i,j}^{\alpha}(t)$ can be controlled by external fields.

D. Time Evolution of Quantum Computers

A quantum algorithm is a sequence of unitary transformations that change the state vector of the quantum computer. Physically, this change corresponds to the time evolution of a quantum system. In general, the quantum system has non-negative probabilities $p_0, p_1, \ldots, p_{2^L-1}$ for being in the states $|0\rangle, \ldots, |2^L - 1\rangle$. Then, the state of the quantum system is described by the density matrix [46, 48]

$$\rho(t) = \sum_{i=1}^{2^L-1} |i\rangle p_i(t) \langle i|, \tag{22}$$

where $p_i(t=0) = p_i$ and the expectation values of physical quantities are given by $\langle A(t)\rangle = \mathrm{Tr}\rho(t)A$. The time evolution of the density matrix (22) trivially follows from the time evolution of each of the pure states $|i\rangle$. Therefore, it is sufficient to focus on methods to compute the time evolution of a pure quantum state. In practice, a calculation of the time evolution of the density matrix takes 2^L times more CPU time than the same calculation for a single pure state.

The time evolution of a pure state of the quantum system is given by the solution of the time-dependent Schrödinger equation [46–48]

$$i\frac{\partial}{\partial t}|\Phi(t)\rangle = H(t)|\Phi(t)\rangle. \tag{23}$$

The solution of Eq. (23) can be written as [46–48]

$$|\Phi(t+\tau)\rangle = U(t+\tau,t)|\Phi(t)\rangle = \exp_+\left(-i\int_t^{t+\tau} H(u)du\right)|\Phi(t)\rangle, \tag{24}$$

where τ denotes the time step. The propagator $U(t+\tau,t) = \exp_+\left(-i\int_t^{t+\tau} H(u)du\right)$ is a unitary matrix that transforms (that is, rotates) the state $|\Phi(t\rangle)$ into the state $|\Phi(t+\tau)\rangle$. The time-ordered matrix exponential $\exp_+\left(-i\int_t^{t+\tau} H(u)du\right)$ is formally defined by

$$\begin{aligned}\exp_+\left(-i\int_t^{t+\tau} H(u)du\right) = 1 \;+\; (-i)\int_t^{t+\tau} du_1\, H(u_1) + (-i)^2\int_t^{t+\tau} du_1\int_t^{u_1} du_2 H(u_1)H(u_2) \\ +\; (-i)^3\int_t^{t+\tau} du_1\int_t^{u_1} du_2\int_t^{u_2} du_3 H(u_1)H(u_2)H(u_3) + \ldots.\end{aligned} \tag{25}$$

For computational purposes, a naive truncation of the series (25) would be fairly useless because it would yield a nonunitary approximation to $U(t+\tau,t)$ and this would violate one of the basic axioms of quantum theory.

In practice, the time dependence of the external fields can always be regarded as piece-wise constant in time. Hence we divide the interval $[t,t+\tau]$ into n smaller intervals $[t_k,t_{k+1}]$ where $t_k = t+\sum_{j=0}^{k}\tau_j$, τ_k denotes the k-th time interval for which $H(t)$ is constant and $\tau = \sum_{j=0}^{n}\tau_j$ ($\tau_0 = 0$). Then we can write [60]

$$U(t+\tau,t) = U(t_n,t_{n-1})U(t_{n-1},t_{n-2})\ldots U(t_1,t_0), \tag{26}$$

where

$$U(t_j,t_{j-1}) = e^{-i\tau_j H(t_{j-1}+\tau_j/2)}. \tag{27}$$

Note that we have not made any assumption about the size of the time intervals $[t_k,t_{k+1}]$ (with respect to the relevant energy scales). These formal manipulations show that a numerical solution of the time-dependent Schrödinger equation for the time dependent model (21) boils down to the calculation of matrix exponentials of the form (27). For $t_{j-1} \le t < t_j$, the Hamiltonian that appears in (27) does not change with time. Hence we can use the shorthand $U(t) = e^{-itH}$, keeping in mind that in actual applications, both t and H may depend on the particular time interval $[t_{j-1},t_j]$.

In general, there are two closely related, strategies to construct algorithms to solve equations such as the time-dependent Schrödinger equation [61]. The traditional approach is to discretize (with an increasing level of sophistication) the derivative with respect to time. A fundamental difficulty with this approach is that it is hard to construct algorithms that preserve the essential character of quantum dynamical time evolution, namely the fact that it is a *unitary transformation* of the state of the quantum system. In particular, in the context of quantum computation it is more natural to consider algorithms that yield a unitary time evolution by construction. In this chapter we review

only methods that are based on the second strategy [61, 62], namely, to approximate the formally exact solution, that is the matrix exponential $U(t) = e^{-itH}$ by some unitary time evolution matrix that is also unitary (to machine precision) $\widetilde{U}(t)$.

If the approximation $\widetilde{U}(t)$ is itself an orthogonal transformation, then $\|\widetilde{U}(t)\| = 1$ where $\|X\|$ denotes the 2-norm of a vector or matrix X [55, 56, 58, 63]. This implies that $\|\widetilde{\mathbf{\Phi}}(t)\| = \|\widetilde{U}(t)\mathbf{\Phi}(0)\| = \|\mathbf{\Phi}(0)\|$, for an arbitrary initial condition $\mathbf{\Phi}(0)$ and for all times t. Hence the time integration algorithm defined by $\widetilde{U}(t)$ is unconditionally stable by construction [61, 62]. In other words, the numerical stability of the algorithm does not depend on the time step t that is used. In Sections III we review and compare different algorithms to compute $U(t) = e^{-itH}$.

E. Single-qubit Operations

The simplest, single-qubit operation on qubit j changes the phase of the amplitudes, depending on whether qubit j is 0 or 1. In terms of spins, the Hamiltonian that performs this operation is

$$H = -h_j^z S_j^z, \tag{28}$$

and is clearly a special case of the ideal quantum computer model (20). Taking $j = 1$ as an example, the unitary transformation

$$U = e^{-itH} = e^{ith_1^z S_1^z}, \tag{29}$$

changes the state (15) into

$$\begin{aligned} |\Phi'\rangle = U|\Phi\rangle &= e^{i\phi}a(\uparrow\uparrow\ldots\uparrow)|\uparrow\uparrow\ldots\uparrow\rangle + e^{-i\phi}a(\downarrow\uparrow\ldots\uparrow)|\downarrow\uparrow\ldots\uparrow\rangle + \ldots \\ &\quad + e^{i\phi}a(\uparrow\downarrow\ldots\downarrow)|\uparrow\downarrow\ldots\downarrow\rangle + e^{-i\phi}a(\downarrow\downarrow\ldots\downarrow)|\downarrow\downarrow\ldots\downarrow\rangle, \\ &= e^{i\phi}\left[a(\uparrow\uparrow\ldots\uparrow)|\uparrow\uparrow\ldots\uparrow\rangle + e^{-2i\phi}a(\downarrow\uparrow\ldots\uparrow)|\downarrow\uparrow\ldots\uparrow\rangle + \ldots\right. \\ &\quad \left. + a(\uparrow\downarrow\ldots\downarrow)|\uparrow\downarrow\ldots\downarrow\rangle + e^{-2i\phi}a(\downarrow\downarrow\ldots\downarrow)|\downarrow\downarrow\ldots\downarrow\rangle\right], \\ &= e^{i\phi}\left[a_0|0\rangle + e^{-2i\phi}a_1|1\rangle + \ldots + a_{2^L-2}|2^L-2\rangle + e^{-2i\phi}a_{2^L-1}|2^L-1\rangle\right], \end{aligned} \tag{30}$$

where the phase shift ϕ is given by $\phi = th_1^z/2$, and we used the fact that in the (spin up, spin down)-representation, S_1^z is a diagonal matrix with eigenvalues (1/2,-1/2). From Eq. (30) it is clear that by a proper choice of the time t and external field h_1^z we can perform any phase shift operation. To obtain the last line of Eq. (30), a global phase factor has been extracted from all the amplitudes. This manipulation does not alter the outcome of the calculation as it eventually drops out in the calculation of expectation values [46–48]. We will often use this trick to simplify the expressions, dropping global phase factors whenever possible.

The phase shift operation $Z_j = e^{i\phi S_j^z}$ with the $\pi/2$ rotations (13) allows us to compose any single qubit operation. For example, using the algebraic properties of the spin operators, it follows that

$$e^{i\phi S^x} = e^{-i\pi S^y/2}e^{i\phi S^z}e^{i\pi S^y/2}, \tag{31}$$

showing that we can readily construct a $\pi/2$ rotation about the z-axis by combining rotations about the x and y-axis.

Some simple examples may help to understand the various operations. For a two-qubit quantum computer we have $|a\rangle = a_0|00\rangle + a_1|01\rangle + a_2|10\rangle + a_3|11\rangle$ and

$$X_1\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \equiv \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & i & 0 & 0 \\ i & 1 & 0 & 0 \\ 0 & 0 & 1 & i \\ 0 & 0 & i & 1 \end{pmatrix}\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}, \tag{32}$$

where X_1 denotes a rotation of qubit 1 by $\pi/2$ about the x-axis. For example $X_1|10\rangle = (|10\rangle + i|11\rangle)/\sqrt{2}$ and $\overline{X}_1|10\rangle = (|10\rangle - i|11\rangle)/\sqrt{2}$. Using the same labeling of states as in (32), we have

TABLE I: Input and output states and the corresponding expectation values (Q_1^z,Q_2^z) of the qubits for the CNOT operation.

Input state	Q_2^z	Q_1^z	Output state	Q_2^z	Q_1^z
$\|00\rangle$	0	0	$\|00\rangle$	0	0
$\|01\rangle$	1	0	$\|11\rangle$	1	1
$\|10\rangle$	0	1	$\|10\rangle$	1	0
$\|11\rangle$	1	1	$\|01\rangle$	0	1

$$Y_2 \equiv \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix}, \tag{33}$$

for example, $Y_2|10\rangle = (|00\rangle + |10\rangle)/\sqrt{2}$ and $\overline{Y}_2|10\rangle = (-|00\rangle + |11\rangle)/\sqrt{2}$.

For later use we introduce the symbol

$$R_j(\phi) = e^{i\phi/2} e^{-i\phi S_j^z} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}, \tag{34}$$

to represent the **single-qubit phase-shift** operation by a phase ϕ on qubit j. A symbolic representation of $R_j(\phi = \pi/k)$ is given in Fig. 1a.

F. Two-qubit Operations

As explained previously, the single qubit operations and the CNOT gate constitute a universal set of gates [9]. The CNOT gate is defined by its action on the computational basis states, as shown in Table I. To simplify the notation, we consider only the two relevant qubits and call them qubit 1 and 2. The CNOT gate flips the second qubit if the state of the first qubit is $|1\rangle$, that is, the first qubit acts as a control qubit for the second one. As shown in Fig. 1b, the CNOT gate can be symbolically represented by a vertical line connecting a dot (control bit) and a cross (target bit). On the ideal quantum computer model (20), the CNOT gate can be implemented by a combination of single-qubit operations and a controlled phase shift operation. In matrix notation, the CNOT operation (see Table I) can be written as

$$\text{CNOT} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \equiv C_{21} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}. \tag{35}$$

Our goal is to show that up to an irrelevant global phase factor, CNOT $= \overline{Y}_2 I_{21}(\pi) Y_2$ where

$$I_{21}(\phi) = e^{i\phi(2S_1^z S_2^z - S_1^z - S_2^z)/2} = \begin{pmatrix} e^{-i\phi/4} & 0 & 0 & 0 \\ 0 & e^{-i\phi/4} & 0 & 0 \\ 0 & 0 & e^{-i\phi/4} & 0 \\ 0 & 0 & 0 & e^{3i\phi/4} \end{pmatrix}. \tag{36}$$

It is easy to see that

$$I_{21}(\phi) = e^{-it(-JS_1^z S_2^z - hS_1^z - hS_2^z)}, \tag{37}$$

where $h = -J/2$ and $Jt = \phi$. Eq. (37) shows that physically the phase-shift operation (36) corresponds to the time evolution of a quantum spin system described by the Ising model.

Using the properties of the $S = 1/2$ matrices, it is easy to show that

$$\mathrm{CNOT} = \overline{Y}_2 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} Y_2 = e^{i\pi/4} \overline{Y}_2 I_{21}(\pi) Y_2. \tag{38}$$

This completes the construction of the set of universal gates that can be implemented on the ideal Ising-model quantum computer (20).

The CNOT operation contains a special case of the **controlled phase shift operation** $R_{ji}(\phi)$. The latter appears in several of the quantum algorithms that are discussed later and therefore it is appropriate to introduce it here. The controlled phase shift operation on qubits 1 and 2 reads

$$R_{21}(\phi) \;=\; e^{i\phi/4} I_{21}(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{pmatrix}. \tag{39}$$

Graphically, the controlled phase shift $R_{ij}(\phi = \pi/k)$ is represented by a vertical line connecting a dot (control bit) and a box denoting a single qubit phase shift by π/k (see Fig. 1c).

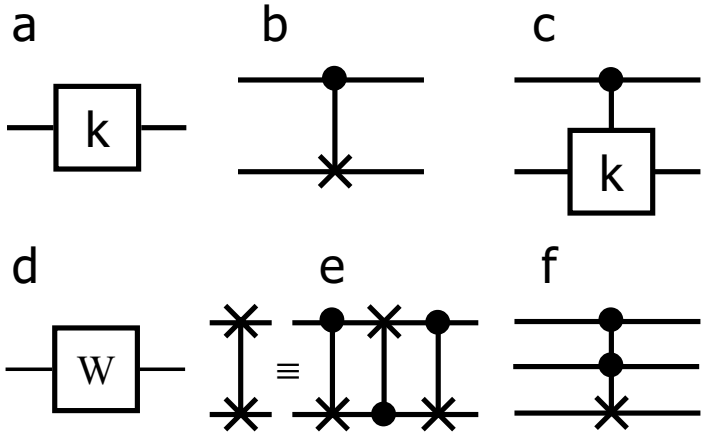

FIG. 1: Graphical representation of some of the basic gates used in quantum computation; (a) single qubit phase shift $R_j(\phi = \pi/k)$ (see Eq. 34); (b) CNOT gate (see Eq. 38); (c) controlled phase shift by $R_{ji}(\phi = \pi/k)$ (see Eq. 39); (d) Walsh-Hadamard gate (see Eq. 76); (e) SWAP gate (see Eq. 78); (f) Toffoli gate (see Eq. 83). The horizontal lines denote the qubits involved in the quantum operations. The dots and crosses denote the control and target bits, respectively.

III. NUMERICAL METHODS

We start by discussing aspects that are common to all numerical algorithms discussed in this review such as techniques to let (combinations of) spin-1/2 operators act on a state. Then we review four different algorithms to compute the time evolution $\widetilde{U}(t)|\Phi\rangle = e^{-itH}|\Phi\rangle$. We start by a brief exposition of the full-diagonalization approach. Then we describe the Chebyshev polynomial algorithm, the short-iterative Lanczos procedure, and several algorithms based on Suzuki-Trotter product formulas.

In numerical work where the use of floating-point arithmetic (e.g. 13-15 digit precision) is necessary, there always will be errors due to the finite precision, rounding and the like. In this chapter we use the term **numerically exact** to indicate that the results are correct up to an error that vanishes with the number of digits that are used to perform the calculation. For example, the value of $\arccos(-1) = \pi$ is numerically exact.

A. General Aspects

As explained previously, the state of a spin-1/2 system with P pairs of L interacting spins or, equivalently, the state of a L-qubit quantum computer, is represented by a complex-valued vector of length $D = 2^L$. In quantum mechanics matrices operating on a vector of length D have dimension $D \times D$. Matrix elements of the Hamiltonian (21) can be complex numbers. Hence we need 2^{2L+4} bytes to store the full matrix. For instance, in the case of a 10-qubit quantum computer we need 16 Mb (16 kb) of memory to store the Hamiltonian (state) which poses no problem for present-day PCs. However, to store the Hamiltonian (state) of the $L = 20$ system, a computer should have 16 Tb (16Mb) of memory. This simple example clearly shows that any simulation scheme that requires storage for the matrix representing the Hamiltonian will be of (very) limited use. Thus, it is necessary to consider techniques to avoid the use of large matrices. Fortunately, for the spin systems considered in this review it is rather straightforward to organize the calculations such that we need only storage for $\mathcal{O}(PL)$ matrices of dimension 2×2 and 4×4.

We first consider the operations $|\Phi'\rangle = S_j^\alpha |\Phi\rangle$. More specifically, let us denote

$$|\Phi\rangle = \quad a(\uparrow\uparrow\ldots\uparrow)|\uparrow\uparrow\ldots\uparrow\rangle + a(\downarrow\uparrow\ldots\uparrow)|\downarrow\uparrow\ldots\uparrow\rangle + \ldots + a(\uparrow\downarrow\ldots\downarrow)|\uparrow\downarrow\ldots\downarrow\rangle + a(\downarrow\downarrow\ldots\downarrow)|\downarrow\downarrow\ldots\downarrow\rangle, \quad (40)$$

and

$$\begin{aligned} |\Phi'\rangle &= S_j^\alpha |\Phi\rangle \\ &= a'(\uparrow\uparrow\ldots\uparrow)|\uparrow\uparrow\ldots\uparrow\rangle + a'(\downarrow\uparrow\ldots\uparrow)|\downarrow\uparrow\ldots\uparrow\rangle + \ldots + a'(\uparrow\downarrow\ldots\downarrow)|\uparrow\downarrow\ldots\downarrow\rangle + a'(\downarrow\downarrow\ldots\downarrow)|\downarrow\downarrow\ldots\downarrow\rangle. \quad (41) \end{aligned}$$

From Eq. (3), it follows that S_j^z is a diagonal matrix in the representation that we use to store the vector $|\Phi\rangle$. Thus we obtain $|\Phi'\rangle$ by reversing the sign of all coefficients of $|\Phi\rangle$ for which the jth bit of their index is one. In terms of amplitudes, we have

$$\begin{aligned} a'(*\ldots*0*\ldots*) &= +\frac{1}{2}\, a(*\ldots*0*\ldots*) \\ a'(*\ldots*1*\ldots*) &= -\frac{1}{2}\, a(*\ldots*1*\ldots*), \end{aligned} \quad (42)$$

where we use the asterisk to indicate that the bits on the corresponding position are the same. From Eq. (3), it follows that S_j^x interchanges states up and down. It immediately follows that the elements of $|\Phi'\rangle$ are obtained by swapping pairs of elements of $|\Phi\rangle$. We have

$$\begin{aligned} a'(*\ldots*0*\ldots*) &= +\frac{1}{2}\, a(*\ldots*1*\ldots*) \\ a'(*\ldots*1*\ldots*) &= +\frac{1}{2}\, a(*\ldots*0*\ldots*). \end{aligned} \quad (43)$$

In Eq. (43), the bit strings on the left- and right-hand side are identical except for the jth bit. In the case of $|\Phi'\rangle = S_j^y |\Phi\rangle$, a similar argument shows that

$$\begin{aligned} a'(*\ldots*0*\ldots*) &= -\frac{i}{2}\, a(*\ldots*1*\ldots*) \\ a'(*\ldots*1*\ldots*) &= +\frac{i}{2}\, a(*\ldots*0*\ldots*). \end{aligned} \quad (44)$$

Note that all these operations can be done *in place*, that is, without using another vector of length 2^L. The three operations discussed here are sufficient to construct any algorithm to simulate the quantum dynamics of model (21). However, for reasons of computational efficiency, it is advantageous to extend the repertoire of elementary operations a little.

The two-spin operation $|\Phi''\rangle = S_j^\alpha S_k^\alpha |\Phi\rangle$ can be implemented in at least three different ways. In the discussion that follows, we exclude the trivial case where $j = k$. One obvious method would be to write it as two single-spin operations of the kind discussed previously: $|\Phi'\rangle = S_k^\alpha |\Phi\rangle$ followed by $|\Phi''\rangle = S_j^\alpha |\Phi'\rangle$. For a single pair (j, k), this is an efficient method. However, if there are many pairs, as in the case of Hamiltonian (21), we have either to recalculate

$S_k^\alpha|\Phi\rangle$ several times, or we have to allocate extra storage to hold $S_k^\alpha|\Phi\rangle$ for $k = 1, \ldots, L$. In both cases, we use extra, costly resources.

In the second approach, we work out analytically how the amplitudes change if we apply $S_j^\alpha S_k^\alpha$ to $|\Phi\rangle$. If $\alpha = z$, we have to reverse the sign of the amplitude only if the jth and kth bit of the amplitude index are different. We have

$$\begin{aligned}
a''(*\ldots*0*\ldots*0*\ldots*) &= +\frac{1}{4}\,a(*\ldots*0*\ldots*0*\ldots*)\\
a''(*\ldots*1*\ldots*0*\ldots*) &= -\frac{1}{4}\,a(*\ldots*1*\ldots*0*\ldots*)\\
a''(*\ldots*0*\ldots*1*\ldots*) &= -\frac{1}{4}\,a(*\ldots*0*\ldots*1*\ldots*)\\
a''(*\ldots*1*\ldots*1*\ldots*) &= +\frac{1}{4}\,a(*\ldots*1*\ldots*1*\ldots*).
\end{aligned} \tag{45}$$

If $\alpha = x$, we interchange quadruples of amplitudes according to the following, rather obvious, rules:

$$\begin{aligned}
a''(*\ldots*0*\ldots*0*\ldots*) &= \frac{1}{4}\,a(*\ldots*1*\ldots*1*\ldots*)\\
a''(*\ldots*1*\ldots*0*\ldots*) &= \frac{1}{4}\,a(*\ldots*0*\ldots*1*\ldots*)\\
a''(*\ldots*0*\ldots*1*\ldots*) &= \frac{1}{4}\,a(*\ldots*1*\ldots*0*\ldots*)\\
a''(*\ldots*1*\ldots*1*\ldots*) &= \frac{1}{4}\,a(*\ldots*0*\ldots*0*\ldots*).
\end{aligned} \tag{46}$$

We leave the case $\alpha = y$ as an exercise for the reader. Clearly, this approach does not require additional storage or calculations to compute intermediate results.

The third method relies on the observation that for $\alpha = x, y$ and $j \neq k$

$$S_j^\alpha S_k^\alpha|\Phi\rangle = R_j^\alpha R_k^\alpha S_j^z S_k^z (R_j^\alpha)^\dagger (R_k^\alpha)^\dagger|\Phi\rangle, \tag{47}$$

where R_j^α is a rotation that transforms S_j^z into S_j^α and that the calculation of $S_j^z S_k^z|\Phi\rangle$ can be done very efficiently. From Eq. (9) it follows that $Y_j S_j^x \overline{Y}_j = S_j^z$ and $\overline{X}_j S_j^y X_j = S_j^z$. Hence it follows that $R_j^x = \overline{Y}_j$ and $R_j^y = X_j$. From Eq. (47), it is clear that we have to determine the rules only to compute $|\Phi'\rangle = (R_j^\alpha)^\dagger|\Phi\rangle$ for $\alpha = x, y$. These rules follow directly from the matrix representations (13) of X_j and Y_j. For instance, in the case of $|\Phi'\rangle = (R_j^x)^\dagger|\Phi\rangle$, we have

$$\begin{aligned}
a'(*\ldots*0*\ldots*) &= +\frac{1}{\sqrt{2}}[a(*\ldots*0*\ldots*) - a(*\ldots*1*\ldots*)]\\
a'(*\ldots*1*\ldots*) &= +\frac{1}{\sqrt{2}}[a(*\ldots*0*\ldots*) + a(*\ldots*1*\ldots*)],
\end{aligned} \tag{48}$$

and similar expressions hold for the other cases.

All the single- and two-spin operations discussed here can be carried out in $\mathcal{O}(2^L)$ floating-point operations. The full diagonalization method, the Chebyshev and short-iterative Lanczos algorithm to be discussed later require a procedure that computes $|\Phi'\rangle = H|\Phi\rangle$. Given the parameters of H [see Eq. (21)], it is straightforward to combine the single- and two-spin operations that were described to perform the operation $|\Phi'\rangle = H|\Phi\rangle$.

The single- and two-qubit operations described before suffice to implement any unitary transformation on the state vector. As a matter of fact, the simple rules described are all that we need to simulate ideal quantum computers on a conventional computer. In Section V we discuss this topic in more detail. For now, we continue with reviewing numerical techniques for computing the time evolution e^{-itH} of a physical system described by a Hamiltonian H.

B. Full Diagonalization Approach

As H is a $D \times D$ Hermitian matrix, it has a complete set of eigenvectors and real-valued eigenvalues [55, 56, 58, 63]. Let us denote the diagonal matrix of all these eigenvalues by Λ and the unitary matrix of all these eigenvectors by V.

Then we have $V^\dagger H V = \Lambda$ and $U(t) = e^{-itH} = V e^{-it\Lambda} V^\dagger$. In other words, once we know Λ and V, $U(t)$ is obtained by simple matrix multiplication. Thus, the most straightforward method to compute $U(t) = e^{-itH} = V e^{-it\Lambda} V^\dagger$ is to use standard linear-algebra algorithms to diagonalize the matrix H. An appealing feature of this approach is that the most complicated part of the algorithm, the diagonalization of the matrix H, is relegated to well-developed linear algebra packages. Memory and CPU time of the standard diagonalization algorithms scale as D^2 and D^3, respectively [55, 56, 63].

The matrix elements of H can be computed by repeated use of the $|\Phi'\rangle = H|\Phi\rangle$ procedure. If we set $|\Phi\rangle = (1, 0, \ldots, 0)^T$, then $|\Phi'\rangle = H|\Phi\rangle$ gives us the first column of the matrix H. The same calculation for $|\Phi\rangle = (0, 1, \ldots, 0)^T$ yields the second column, and so on.

The application of the full-diagonalization approach is limited by the memory and CPU resources it requires. The former is often the most limiting factor (see the prior discussion), in spite of the fact that full diagonalization takes $\mathcal{O}(2^{3L})$ floating-point operations. In practice, solving the time-dependent Schrödinger equation for a system of 12-13 spins for many pairs (H,t) (which is necessary if there are time-dependent fields) is many orders of magnitude more expensive in terms of computational resources than solving the same problem by the methods discussed later. Nevertheless, any toolbox for simulating quantum spin dynamics should include code that is based on the full diagonalization approach. For all but the most trivial problems, it is an essential tool to validate the correctness of other algorithms that solve the time-dependent Schrödinger equation.

C. Chebyshev Polynomial Algorithm

The basic idea of this approach is to make use of a numerically exact polynomial approximation to the matrix exponential $U(t) = e^{-itH}$ [64–70]. The first step in the algorithm is to "normalize" the matrix H such that its (unknown) eigenvalues lie in the interval $[-1, 1]$. In general, the eigenvalues E_j of the Hermitian matrix H are real numbers in the interval $[-\|H\|, \|H\|]$ where $\|H\| \equiv \max_j |E_j|$ is the 2-norm of the matrix H [55, 56]. Unless we already solved the eigenvalue problem of H, we usually do not know the exact value of $\|H\|$. For the Hamiltonian (21), it is easy to compute an upper bound to $\|H\|$ by repeated use of the triangle inequality

$$\|X + Y\| \leq \|X\| + \|Y\|, \tag{49}$$

and the elementary bounds $\|S_j^\alpha\| = 1/2$ and $\|S_j^\alpha S_k^\alpha\| = 1/4$. For fixed t, we can write $H = H(t)$, and we find [70]

$$\|H\| \leq \|H\|_b \equiv \frac{1}{4} \sum_{i,j=1}^{L} \sum_{\alpha=x,y,z} |J_{i,j}^\alpha| + \frac{1}{2} \sum_{i=1}^{L} \sum_{\alpha=x,y,z} |h_i^\alpha|. \tag{50}$$

Maximum efficiency of the Chebyshev polynomial algorithm is obtained if we know the exact value $\|H\|$ (see later). Thus, if we have more specific information about the model parameters $J_{i,j}^\alpha$ and h_i^α, it should be used to improve the bound on $\|H\|$. By construction, the eigenvalues of $\widehat{H} \equiv H/\|H\|_b$ all lie in the interval $[-1, 1]$.

Expanding the initial state $\Phi(0)$ in the (unknown) eigenvectors $|E_j\rangle$ of H, we have

$$|\Phi(t)\rangle = e^{-itH}|\Phi(0)\rangle = e^{-i\widehat{t}\widehat{H}}|\Phi(0)\rangle = \sum_j e^{-i\widehat{t}\widehat{E}_j}|E_j\rangle\langle E_j|\Phi(0)\rangle, \tag{51}$$

where $\widehat{t} = t\|H\|_b$ and the $\widehat{E}_j = E_j/\|H\|_b$ denote the (unknown) eigenvalues of $\widehat{H}$ (in practice, there is no need to know the eigenvalues and eigenvectors of H explicitly). Next, recall that for $|x| \leq 1$ we have [71]

$$e^{-izx} = J_0(z) + 2\sum_{k=1}^{\infty} (-i)^k J_k(z) T_k(x), \tag{52}$$

where $J_k(z)$ is the Bessel function of integer order k, and $T_k(x) = \cos[k \arccos(x)]$ is the Chebyshev polynomial of the first kind [71]. As $-1 \leq \widehat{E}_j \leq 1$, we can use Eq. (52) to write Eq. (51) as

$$|\Phi(t)\rangle = \left[J_0(-\widehat{t})\mathbb{1} + 2\sum_{k=1}^{\infty} J_k(-\widehat{t})\widehat{T}_k(\widehat{H}) \right] |\Phi(0)\rangle, \tag{53}$$

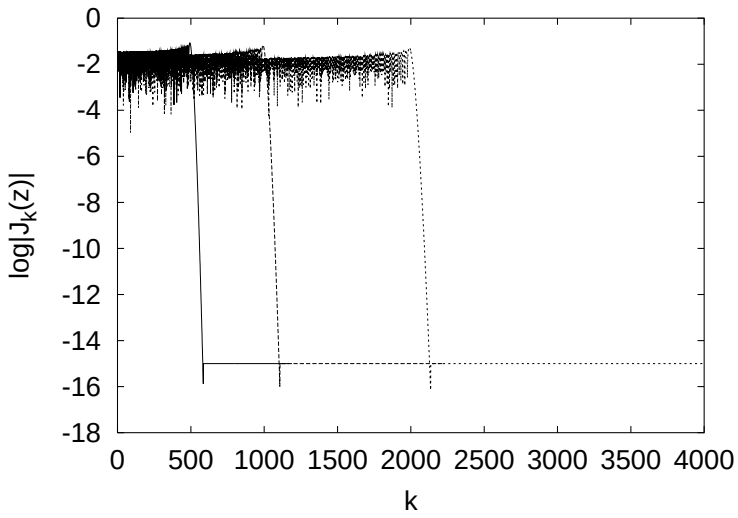

FIG. 2: Dependence of the Bessel function $J_k(z)$ on the order k; solid line: $z = 500$; dashed line: $z = 1000$; dotted line: $z = 2000$.

where $\widehat{T}_k(\widehat{H}) = i^k T_k(\widehat{H}) = i^k \sum_j |E_j\rangle T_k(\widehat{E}_j)\langle E_j|$ is a matrix-valued, modified Chebyshev polynomial that is defined by the recursion relations

$$\widehat{T}_0(\widehat{H})|\Psi\rangle = |\Psi\rangle\,, \quad \widehat{T}_1(\widehat{H})|\Psi\rangle = i\widehat{H}|\Psi\rangle\,, \tag{54}$$

and

$$\widehat{T}_{k+1}(\widehat{H})|\Psi\rangle = 2i\widehat{H}\widehat{T}_k(\widehat{H})|\Psi\rangle + \widehat{T}_{k-1}(\widehat{H})|\Psi\rangle\,, \tag{55}$$

for $k \geq 1$.

For practical purposes the sum in Eq.(53) has to be truncated. We compute the approximation $|\widetilde{\Phi}(t)\rangle = \widetilde{U}(t)|\Phi(0)\rangle$ by keeping the first $K+1$ contributions:

$$|\widetilde{\Phi}(t)\rangle = \left[J_0(-\widehat{t})\mathbb{1} + 2\sum_{k=1}^{K} J_k(-\widehat{t})\widehat{T}_k(\widehat{H})\right]|\Phi(0)\rangle. \tag{56}$$

As $|J_k(-\widehat{z})| < 1$ and $\|\widehat{T}_k(\widehat{H})\| = \|T_k(\widehat{H})\| \leq 1$, all contributions in Eq. (56) are of the same order of magnitude. In contrast, the truncated Taylor series $e^{-itH} \approx \sum_{k=1}^{K}(-itH)^k/k!$ is a sum of many small and large numbers, and a numerical calculation of the sum is known to suffer from severe instabilities [72].

Using the downward recursion relation of the Bessel functions, it is straightforward to calculate the first K Bessel functions to machine precision in $\mathcal{O}(K)$ arithmetic operations [71, 73]. In practice, a calculation of the first 20,000 Bessel functions takes less than 1 second on a Pentium III 600 MHz mobile processor, using 14 - 15 digit arithmetic. To get a feeling for the value of K at which the Chebyshev polynomial expansion might be truncated, it is instructive to plot $J_k(z)$ as a function of k. From Fig.2 it is clear that $|J_k(z)|$ vanishes (very) rapidly if k becomes larger than z. In fact, $|J_k(z)| \leq z^k/2^k k!$ for real z hence $|J_k(z)|$ vanishes exponentially fast as k increases [71]. Thus we can fix the number K by requiring that $|J_k(z)| > \kappa$ for all $k \leq K$. Here κ is a small number that determines the accuracy of the approximation. From Fig.2, it follows that $K \approx z$ and that $|J_k(z = 2000)| < 10^{-10}$ for all $k > z + 100$.

Once we have found the smallest K such that $|J_k(-\widehat{t})| > \kappa$ for all $k \leq K$, there is no point of taking more than K terms in the expansion. Since $\|\widehat{T}_k(\widehat{H})\| \leq 1$, Fig.2 suggests that including such contributions would only add to the noise, and numerical tests show that this is indeed the case. However, taking less than K terms has considerable negative impact on the accuracy of the results. Hence in practice the choice of K is rather limited (such as $K \approx z + 200$ if $z = 2000$ and machine precision is required). Most important, for fixed κ, K increases linearly with $t\|H\|_b$. Therefore, if we can replace the bound $\|H\|_b$ by a sharper one, the value of K can be reduced accordingly and the calculation will take less CPU time.

In a strict sense, the truncated Chebyshev polynomial in Eq. (56) is not a unitary matrix. However, the error $\|\Phi(t)-\widehat{\Phi}(t)\|$ between the exact state $\Phi(t)$ and the truncated Chebyshev polynomial result $\widehat{\Phi}(t)$ vanishes (exponentially) fast as K increases. Therefore, for practical purposes the truncated Chebyshev polynomial can be viewed as an extremely stable time-integration algorithm because it yields an approximation to the exact time evolution operator $U(t) = e^{-itH}$ that is numerically exact. Under these conditions, the Chebyshev algorithm may safely be used repeatedly to perform multiple time steps with a (very) large fixed time step [70].

According to Eq.(56), performing one time step amounts to repeatedly using recursion relation (55) to obtain $\widehat{T}_k(\widehat{H})\Phi(0)$ for $k = 2, \ldots, K$, then multiply the elements of this vector by $J_k(-\widehat{t})$ and add all contributions. This procedure requires storage for two vectors of the same length as Φ and a procedure that returns $H|\Phi\rangle$. Both memory and CPU time of this method are $\mathcal{O}(D = 2^L)$. CPU time increases linearly with the time step t.

D. Short-Iterative Lanczos Algorithm

The short-iterative Lanczos algorithm [66, 74–77] belongs to the class of reduced-order methods that are based on the approximation of a matrix by its projection onto a (much) smaller subspace. The short-iterative Lanczos algorithm is based on the approximation

$$e^{-itH}|\Psi\rangle \approx \widetilde{U}_N(t) = e^{-itP_NHP_N}|\Psi\rangle, \tag{57}$$

where P_N is the projector on the N-dimensional subspace spanned by the vectors $\{\Psi, H\Psi, \ldots, H^{N-1}\Psi\}$. As $e^{-itP_NHP_N}$ is unitary the method is unconditionally stable.

There are two major steps in the short-iterative Lanczos algorithm. First we calculate $P_NHP_N\Psi$ by generating the orthogonal Lanczos vectors in the usual manner [55, 56]:

$$\begin{aligned}
|\Psi'_0\rangle &= |\Psi\rangle, \\
|\Psi_0\rangle &= |\Psi'_0\rangle/\sqrt{\langle\Psi'_0|\Psi'_0\rangle}, \\
|\Psi'_1\rangle &= H|\Psi_0\rangle - |\Psi_0\rangle\langle\Psi_0|H|\Psi_0\rangle, \\
|\Psi_1\rangle &= |\Psi'_1\rangle/\sqrt{\langle\Psi'_1|\Psi'_1\rangle},
\end{aligned} \tag{58}$$

and

$$\begin{aligned}
|\Psi'_k\rangle &= H|\Psi_{k-1}\rangle - |\Psi_{k-1}\rangle\langle\Psi_{k-1}|H|\Psi_{k-1}\rangle - |\Psi_{k-2}\rangle\langle\Psi_{k-2}|H|\Psi_{k-2}\rangle, \\
|\Psi_k\rangle &= |\Psi'_k\rangle/\sqrt{\langle\Psi'_k|\Psi'_k\rangle},
\end{aligned} \tag{59}$$

for $1 \le k \le N$. By construction, we have $\langle\Psi_k|\Psi_{k'}\rangle = \delta_{k,k'}$, and the projection with $P_N = \sum_{k=1}^{N}|\Psi_k\rangle\langle\Psi_k|$ yields a $N \times N$ tri-diagonal matrix P_NHP_N. The second step in the short-iterative Lanczos algorithm is to diagonalize this tridiagonal matrix and use the resulting eigenvalues and eigenvectors to compute $e^{-itP_NHP_N}|\Psi\rangle$. The latter is done in exactly the same manner as in the case of the full diagonalization method.

The short-iterative Lanczos algorithm requires storage for all the eigenvectors and/or the N Lanczos vectors $|\Psi_k\rangle$ of the $N \times N$ matrix P_NHP_N. Thus, for this method to compete with the full diagonalization method, we require $N \ll D = 2^L$. The accuracy of this algorithm depends both on the order N and the state $|\Psi\rangle$ [66, 74, 75]. With infinite-precision arithmetic, $e^{-itH}|\Psi\rangle = \lim_{N\to\infty} e^{-itP_NHP_N}|\Psi\rangle$, but in practice, the loss of orthogonality during the Lanczos procedure limits the order N and the time step t that can be used without introducing spurious eigenvalues [55, 56]. The low-order short-iterative Lanczos algorithm may work well if $|\Psi\rangle$ contains contributions from eigenstates of H that are close in energy. However, if $|\Psi\rangle$ contains contributions from many eigenstates of H with very different energies, it is unlikely that all these eigenvalues will be present in P_NHP_N, and the approximation $\widetilde{U}_N(t)$ may be rather poor unless N is sufficiently large. Memory and CPU time (for one time step) of the short-iterative Lanczos algorithm scale as D and N^2D, respectively. In general, N increases with t in a nontrivial, problem-dependent manner that seems difficult to determine in advance.

E. Suzuki-Trotter Product-Formula Algorithms

A systematic approach to construct unitary approximations to unitary matrix exponentials is to make use of the Lie-Trotter-Suzuki product-formula [78, 79]

$$U(t) = e^{-itH} = e^{-it(H_1+\ldots+H_K)} = \lim_{m\to\infty}\left(\prod_{k=1}^{K} e^{-itH_k/m}\right)^m, \tag{60}$$

and generalizations thereof [80–82]. Expression (60) suggests that

$$\widetilde{U}_1(t) = e^{-itH_1}\ldots e^{-itH_K}, \tag{61}$$

might be a good approximation to $U(t)$ if t is sufficiently small. From Eq. (61), it follows that the Taylor series of $U(t)$ and $\widetilde{U}_1(t)$ are identical up to first order in t. We call $\widetilde{U}_1(t)$ a first-order approximation to $U(t)$. If all the H_i in Eq. (61) are Hermitian, then $\widetilde{U}_1(t)$ is unitary by construction, and a numerical algorithm based on (61) will be unconditionally stable. For unitary matrices $U(t)$ and $\widetilde{U}_1(t)$, it can be shown that [62]

$$\|U(t) - \widetilde{U}_1(t)\| \leq \frac{t^2}{2}\sum_{i<j}\|[H_i, H_j]\|, \tag{62}$$

suggesting that $\widetilde{U}_1(t)$ may be a good aproximation if we advance the state of the quantum system by small time steps t such that $t\|H\| \ll 1$. Note that this is the situation of interest if there are external time-dependent fields.

The Suzuki-Trotter product-formula approach provides a simple, systematic framework to improve the accuracy of the approximation to $U(\tau)$ with marginal programming effort and without changing its fundamental properties. For example, the matrix

$$\widetilde{U}_2(t) = \widetilde{U}_1^\dagger(-t/2)\widetilde{U}_1(t/2) = e^{-itH_K/2}\ldots e^{-itH_1/2}e^{-itH_1/2}\ldots e^{-itH_K/2}, \tag{63}$$

is a second-order approximation to $\widetilde{U}(t)$ [80–82]. If $\widetilde{U}_1(t)$ is unitary, so is $\widetilde{U}_2(t)$. For unitary $\widetilde{U}_2(t)$, we have [62]

$$\|U(t) - \widetilde{U}_2(t)\| \leq c_2 t^2, \tag{64}$$

where c_2 is a positive constant [62].

Higher-order approximations based on $\widetilde{U}_2(t)$ [or $\widetilde{U}_1(t)$] can be constructed by using Suzuki's fractal decomposition [80, 82]. A particularly useful fourth-order approximation is given by [80, 82]

$$\widetilde{U}_4(t) = \widetilde{U}_2(at)\widetilde{U}_2(at)\widetilde{U}_2((1-4a)t)\widetilde{U}_2(at)\widetilde{U}_2(at), \tag{65}$$

where $a = 1/(4-4^{1/3})$. As before, if $\widetilde{U}_2(t)$ is unitary, so is $\widetilde{U}_4(t)$ and we have

$$\|U(t) - \widetilde{U}_4(t)\| \leq c_4 t^4, \tag{66}$$

where c_4 is a positive constant. The rigorous error bounds (62), (64) and (66) suggest that for sufficiently small t, the numerical error $\|U(t)\Psi - \widetilde{U}_n(t)\Psi\|$ vanishes as t^n. If this behavior is not observed, there is a fair chance that there is at least one mistake in the computer program that implements $\widetilde{U}_n(t)$. For a fixed accuracy, memory and CPU time of an $\widetilde{U}_n(t)$ algorithm scale as $\mathcal{O}(D)$ and $\mathcal{O}(t^{(1+1/n)}D)$, respectively. The approximations (63) and (65) have proven to be very useful for a wide range of different applications [62, 79, 81, 83–94].

In practice, an efficient implementation of the first-order scheme is all that is needed to build higher-order algorithms (63) and (65). The crucial step that we have to consider now is how to choose the Hermitian H_is such that the matrix exponentials $\exp(-itH_1)$, ..., $\exp(-itH_K)$ can be calculated efficiently.

If there are external time-dependent fields, as in the case of a physical quantum computer, it is expedient to decompose the Hamiltonian into single-spin and two-spin contributions. The rational behind this is that in most cases of interest (NMR,...) the external fields change on a much shorter time scale than the other parameters in the Hamiltonian. As the spin operators with different qubit labels commute, we have

$$\exp\left\{-it\left[-\sum_{j=1}^{L}\sum_{\alpha=x,y,z}h_j^\alpha S_j^\alpha\right]\right\}=\prod_{j=1}^{L}\exp\left[it\sum_{\alpha=x,y,z}h_j^\alpha S_j^\alpha\right]. \tag{67}$$

The jth factor of the product in (67) rotates spin j about the vector $\mathbf{h}_j=(h_j^x,h_j^y,h_j^z)$. Each factor can be worked out analytically, yielding

$$e^{it\mathbf{S}_j\cdot\mathbf{h}_j}=\begin{pmatrix}\cos\frac{th_j}{2}+\frac{ih_j^z}{h_j}\sin\frac{th_j}{2} & \frac{ih_j^x+h_j^y}{h_j}\sin\frac{th_j}{2}\\ \frac{ih_j^x-h_j^y}{h_j}\sin\frac{th_j}{2} & \cos\frac{th_j}{2}-\frac{ih_j^z}{h_j}\sin\frac{th_j}{2}\end{pmatrix}, \tag{68}$$

where $h_j=\|\mathbf{h}_j\|$ is the length of the vector $\mathbf{h}_j$. From Eq. (67) and Eq. (68), we conclude that the time evolution due to single-spin terms can be calculated by performing $\mathcal{O}(LD)$ operations involving 2×2 matrices. Because all matrix elements are non-zero, these single-spin operations are only marginally more complicated than the ones discussed in Section III A.

We now consider two different decompositions of the two-spin terms that can be implemented very efficiently: The original pair-product split-up [79, 95] in which H_j contains all contributions of a particular pair of spins, and an XYZ decomposition in which we break up the Hamiltonian according to the x, y, and z components of the spin operators [93].

The pair-product decomposition is defined by [79, 95]

$$\exp\left\{-it\left[-\sum_{j,k=1}^{L}\sum_{\alpha=x,y,z}J_{j,k}^\alpha S_j^\alpha S_k^\alpha\right]\right\}=\prod_{j,k=1}^{L}\exp\left[it(J_{j,k}^x S_j^x S_k^x+J_{j,k}^y S_j^y S_k^y+J_{j,k}^z S_j^z S_k^z)\right], \tag{69}$$

where the order of the factors in the product over (j,k) is arbitrary. This freedom may be exploited to increase the execution speed by vectorization and/or parallelization techniques. Also in this case the expression of each factor can be worked out analytically and reads

$$e^{it(J_{j,k}^x S_j^x S_k^x+J_{j,k}^y S_j^y S_k^y+J_{j,k}^z S_j^z S_k^z)}=\begin{pmatrix}e^{iat}\cos bt & 0 & 0 & ie^{iat}\sin bt\\ 0 & e^{-iat}\cos ct & ie^{-iat}\sin ct & \\ 0 & ie^{-iat}\sin ct & e^{-iat}\cos ct & \\ ie^{iat}\sin bt & 0 & 0 & e^{iat}\cos bt\end{pmatrix}, \tag{70}$$

where $a=J_{j,k}^z/4$, $b=(J_{j,k}^x-J_{j,k}^y)/4$, and $c=(J_{j,k}^x+J_{j,k}^y)/4$. From Eq. (69) and Eq. (70), it follows that the time evolution due to the spin-spin coupling terms can be calculated by performing $\mathcal{O}(PD)$ operations involving 4×4 matrices. These 4×4 matrix operations are marginally more complicated than the ones discussed in Section III A and therefore we omit further details.

The XYZ decomposition is defined by [93]

$$\exp\left\{-it\left[-\sum_{j,k=1}^{L}\sum_{\alpha=x,y,z}J_{j,k}^\alpha S_j^\alpha S_k^\alpha\right]\right\}=\prod_{\alpha=x,y,z}e^{-itH^\alpha}, \tag{71}$$

where there is no particular order of the factors in Eq. (71) and

$$H^\alpha=-\sum_{j,k=1}^{L}J_{j,k}^\alpha S_j^\alpha S_k^\alpha. \tag{72}$$

The computational basis states are eigenstates of the S_j^z operators. Thus, in this representation e^{-itH^z} is diagonal by construction, and it changes the input state by altering the phase of each of the basis vectors. As H^z is a sum of pair interactions, it is trivial to implement this operation as a sequence of multiplications by 4×4 diagonal matrices. Still working in the same representation, an efficient algorithm that implements the action of e^{-itH^x} (e^{-itH^y}) by using the Y_j (X_j) rotations can be constructed as follows. Writing $Y=\prod_{j=1}^{L}Y_j$ we have

$$e^{-itH^x} = \overline{Y} Y e^{-itH^x} \overline{Y} Y = \overline{Y} \exp\left(it \sum_{j,k=1}^{L} J^x_{j,k} S^z_j S^z_k \right) Y. \tag{73}$$

From Eq. (73), it is clear that the action of e^{-itH^x} can be computed by applying the L single-spin rotations Y, a sequence of multiplications by 4×4 diagonal matrices containing phase shifts, and the L inverse rotations $\overline{Y}$. A similar procedure is used to compute the action of e^{-itH^y}. We have only to replace Y ($J^x_{j,k}$) by $\overline{X}$ ($J^y_{j,k}$). The total arithmetic operation count of the XYZ decomposition (71) is $\mathcal{O}(LD)$. If there is enough memory on the conventional computer to store all the values of $\sum_{j,k=1}^{L} J^\alpha_{j,k} S^z_j S^z_k$ for $\alpha = x, y, z$ and $S^z_j = \pm 1/2$, then the XYZ decomposition runs in $\mathcal{O}(D)$.

F. Comments

All the algorithms just discussed satisfy the basic requirement (unitarity to machine precision) of a valid quantum mechanical time evolution. A general analysis of the strong and weak points of these algorithms is rather difficult: experience shows that the conclusions may change significantly with the particular application. Therefore, we can only consider specific examples and compare algorithms in terms of accuracy, memory, and CPU requirements, and that is what we will do in this subsection.

As an example, we consider a model of two spins $(\mathbf{S}_1, \mathbf{S}_2)$ interacting with a "bath" of $L-2$ spins $(\mathbf{I}_n)$ described by the Hamiltonian [96]

$$H = J_0(\mathbf{S}_1 + \mathbf{S}_2)^2 + \sum_{n=1}^{L-2} J_n \mathbf{I}_n \cdot (\mathbf{S}_1 + \mathbf{S}_2). \tag{74}$$

This model has been used to study decoherence in a two-spin system due to coupling with a bath of spins [96]. The Heisenberg exchange interactions $\{J_n\}$ are assumed to be random, uncorrelated, and small compared with J_0 ($|J_n| \ll |J_0|$). Initially, the two spins $\mathbf{S}_1$ and $\mathbf{S}_2$ are in the state with one spin up and the other spin down. The initial state of the spins $\{\mathbf{I}_n\}$ is assumed to be random. Obviously, Eq. (74) is a special case of the generic Hamiltonian (21).

In Fig. 3, we show simulation results of the time evolution of the magnetization $\langle S^z_1(t) \rangle$ over a large time span, for a spin-bath containing 22 spins ($L = 24$). On a fine time scale (compared with the scale used in Fig.3), the first and second spin oscillate rapidly with a period of order $1/J_0$ (results not shown) [96]. Initially, the amplitude of the magnetization oscillations rapidly decays to zero, then increases again, and then decays to zero very slowly [96]. This is the generic behavior of the magnetization in this class of models [96].

We use the system (74) to compare the different time-integration algorithms. In Table II, we show the error on the final state and the CPU time it took to solve the time-dependent Schrödinger equation. The error is defined as $\|\Psi_*(m) - \Psi_{\mathrm{X}}(m)\|$ where * denotes the algorithm (exact diagonalization or Chebyshev polynomial method) that is used to generate the reference solution and X is one of the other algorithms. It is clear that the short iterative Lanczos method is not competitive for this type of time-dependent Schrödinger problem. The fourth-order pair-approximation is close but is still less efficient than the Chebyshev algorithm. The other Suzuki product-formula algorithms are clearly not competitive. The reason that the pair-approximation is performing fairly well in this case is related to the form of the Hamiltonian (74). The present results support our earlier finding [70] that the numerical simulation of decoherence in spin systems is most efficiently done in a two-step process: The Chebyshev algorithm can be used to make a big leap in time, followed by a Suzuki product-formula algorithm calculation to study the time dependence on a more detailed level.

In Table III, we present a necessarily rough overall assessment of the virtues and limitations of the algorithms discussed in this review. From a general perspective, to increase the confidence in numerical simulation results, it is always good to have several different algorithms to perform the same task.

IV. QUANTUM ALGORITHMS

A quantum algorithm is a sequence of unitary operations that changes the internal state of the quantum computer [9]. Quantum algorithms vary in complexity. In the first subsection, we discuss the elementary gates of a quantum computer. Two of these gates – namely the CNOT gate and the controlled phase shift – have already been discussed in Section II F. These quantum gates are rather simple quantum algorithms that can be used to build quantum

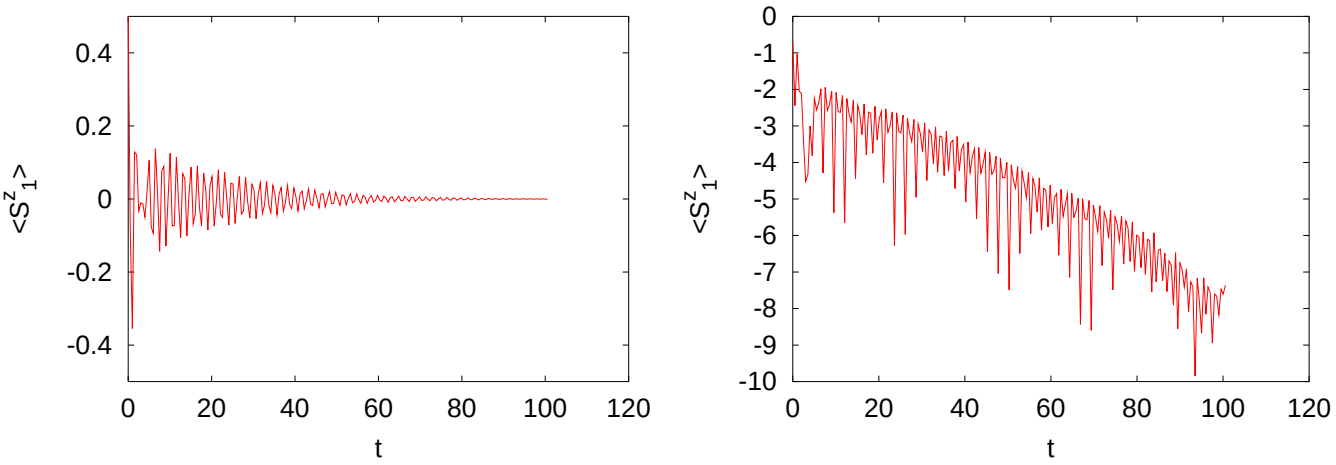

FIG. 3: Left: Magnetization $\langle S_1^z(t)\rangle$ as a function of time as obtained by numerical simulation of two spins interacting with a bath of 22 spins. The parameters of model (74) are $J_0 = 8$ and the J_n are uncorrelated random numbers that satisfy $0 < J_n < 0.4$. Right: $\log|\langle S_1^z(t)\rangle|$ as a function of time showing exponential decay at long times (note that the intrinsic time scale of $S_1^z(t)$ is set by $J_0 = 8$).

TABLE II: Comparison of algorithms to solve the time-dependent Schrödinger equation for model (74). All calculations use a time step $\tau/2\pi = 0.01$. L is the total number of spins, m the number of time steps, and N denotes the number of iterations in the short-iterative Lanczos algorithm. An asterisk in a column indicates that the state obtained by that particular method is used as reference to compare with states obtained by other algorithms. The calculations for $L = 22$ where carried out on a Cray SV1 system. All other calculations were performed on an IBM T40 Notebook with an Intel Centrino (1.3GHz) processor and 512 MB memory.

	Exact Diagonalization	Chebyshev Polynomial	Iterative Lanczos	Suzuki Pair(2)	Suzuki Pair(4)	Suzuki XYZ(2)	Suzuki XYZ(4)
$L = 10$, $m = 400$							
Error (N=5)	*	0.34E-12	0.17E-05	0.23E-03	0.75E-08	0.14E+00	0.53E-04
CPU time [s]	72	0.5	3.5	0.8	3.6	0.6	2.6
$L = 12$, $m = 400$							
Error (N=5)	-	*	0.27E-05	0.27E-03	0.80E-08	0.14E+00	0.55E-04
CPU time [s]	-	2.1	17.3	3.7	18.0	2.5	12.1
$L = 12$, $m = 400$							
Error (N=10)	-	*	0.81E-13	0.27E-03	0.80E-08	0.14E+00	0.55E-04
CPU time [s]	-	2.1	36.2	3.7	18.0	2.5	12.1
$L = 18$, $m = 40$							
Error (N=5)	-	*	0.97E-06	0.90E-04	0.12E-07	0.21E-01	0.94E-0.5
CPU time [s]	-	82	316	74.9	312.5	52.3	239.6
$L = 22$, $m = 8$							
Error (N=5)	-	*	0.40E-06	0.35E-04	0.21E-07	0.57E-02	0.39E-05
CPU time [s]	-	826	1817	402	1510	412	1549

networks of more complicated quantum algorithms such as Grover's database search algorithm and Shor's factoring algorithm These more complicated quantum algorithms are discussed in Subsections IV.B - IV.H. In this Section we discuss only the salient features of the quantum algorithms. Much more detailed information can be found in, for example, references [9, 38, 57, 97–99].

TABLE III: Overall comparison of different integration methods to solve the time-dependent Schrödinger equation. $D = 2^L$ is the dimension of the Hilbert space, T is the time interval of integration, τ is the time step, and N denotes the number of iterations in the short-iterative Lanczos algorithm. An entry MP indicates that the result is numerically exact to machine precision.

	Exact Diagonalization	Chebyshev Polynomial	Iterative Lanczos	Suzuki Pair(2)	Suzuki Pair(4)	Suzuki XYZ(2)	Suzuki XYZ(4)
Memory CPU	$\mathcal{O}(D^2)$ $\mathcal{O}(D^3)$	$\mathcal{O}(D)$ $\mathcal{O}(DT)$	$\mathcal{O}(ND)$ $\mathcal{O}(N^2DT/\tau)$	$\mathcal{O}(D)$ $\mathcal{O}(DT^{3/2})$	$\mathcal{O}(D)$ $\mathcal{O}(DT^{3/4})$	$\mathcal{O}(D)$ $\mathcal{O}(DT^{3/2})$	$\mathcal{O}(D)$ $\mathcal{O}(DT^{3/2})$
Unitary	YES	MP	YES	YES	YES	YES	YES
$H(t)$ changes very often	Inefficient	Slow	Ok	Ok	Ok	Ok	Ok
Overall usefulness	Essential for testing	Very efficient for large T	Limited: N not known in advance	Best choice if $H(t)$ changes rapidly with t	Less accurate but competitive with Chebyshev	Good choice if $H(t)$ changes rapidly with t	Less efficient than Suzuki Pair(4)

A. Elementary Gates

1. Hadamard Gate

The Hadamard gate is a single-qubit gate and is often used to prepare the state of uniform superposition [9]. The Hadamard operation on qubit j is defined by

$$W_j = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{75}$$

For example

$$W_j|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle). \tag{76}$$

In terms of the elementary rotations X and Y, the Hadamard operation on qubit j reads

$$W_j = -iX_j^2\overline{Y}_j = -iY_jX_j^2 = i\overline{X}_j^2\overline{Y}_j = iY_j\overline{X}_j^2. \tag{77}$$

The Hadamard operation can be generalized to an arbitrary number of qubits [9]. The generalized operation is known as the Hadamard transform or as the Walsh-Hadamard transform. This operation consists of L Hadamard gates acting in parallel on L qubits. For example, for $L = 2$, $W_2W_1|00\rangle = (|00\rangle + |01\rangle + |10\rangle + |11\rangle)/2$. The Walsh-Hadamard transform produces a uniform superposition of all basis states. A symbolic representation of the Walsh-Hadamard gate is given in Fig. 1d.

2. Swap Gate

The swap gate interchanges two qubits and is useful in, for example, quantum algorithms that perform Fourier transformations [9]. In matrix notation, the SWAP operation reads

$$\text{SWAP}\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}, \tag{78}$$

showing that the SWAP operation can be decomposed into three CNOT operations. A graphical representation of the swap gate is shown in Fig. 1e).

3. Toffoli Gate

The Toffoli gate is a generalization of the CNOT gate in the sense that it has two control qubits and one target qubit [9, 57]. The target qubit flips if and only if the two control qubits $Q_1^z = Q_2^z = 1$ (see Table IV). Symbolically the Toffoli gate is represented by a vertical line connecting two dots (control bits) and one cross (target bit), as shown in Fig. 1f. There are many ways to decompose the Toffoli gate in one- and two-qubit operations [9]. We discuss two examples.

A quantum network for the first implementation is given in Fig. 4 [9, 57]. It consists of two CNOT gates and three controlled phase shifts.

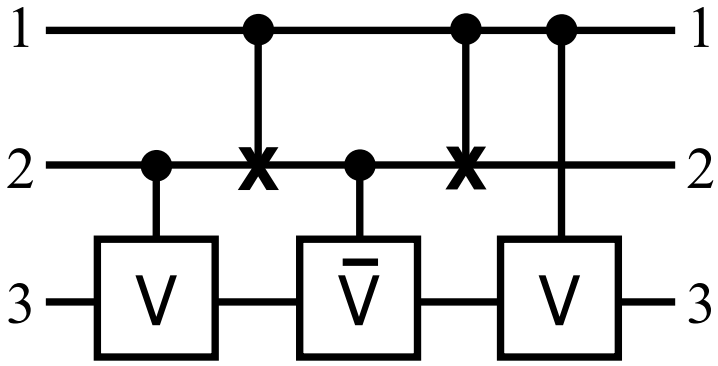

FIG. 4: Quantum network for the Toffoli gate using CNOT gates and controlled phase shifts, which are part of the operations V and $\overline{V}$.

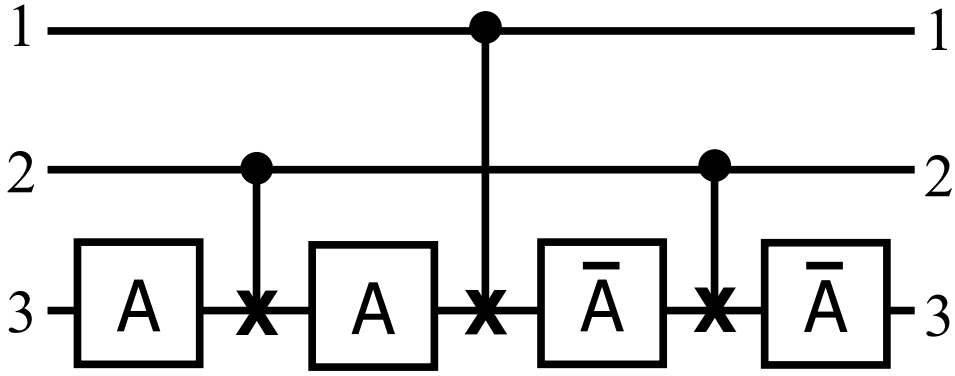

FIG. 5: Quantum network for the Toffoli gate using CNOT gates and single-qubit operation A. A ($\overline{A}$) is a rotation by $\pi/4$ ($-\pi/4$) about the y-axis.

The internal operation of this quantum network is shown in Table IV. The first two columns give all possible combinations of Q_1^z and Q_2^z. The next five columns schematically show the corresponding operations as we move through the network (see Fig. 4) from left to right. The columns labeled "CNOT" show only the value of the second qubit because the first qubit acts as a control bit and does not change. The last column summarizes the operation on the third qubit. From Table IV it immediately follows that operation V has to be constructed such that V^2 flips Q_3 and that $\overline{V}V$ is equal to the identity matrix. These conditions are fulfilled by taking

$$V = \frac{e^{-i\pi/4}}{\sqrt{2}} \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix}. \tag{79}$$

TABLE IV: Internal operation of the quantum network, Fig. 4 representing the Toffoli gate.

Q_1^z	Q_2^z	V	CNOT	$\overline{V}$	CNOT	V	Result
0	0						Q_3^z
0	1	V		$\overline{V}$			$V\overline{V}Q_3^z = Q_3^z$
1	0		1	$\overline{V}$	0	V	$\overline{V}VQ_3^z = Q_3^z$
1	1	V	0		1	V	$V^2Q_3^z = 1 - Q_3^z$

TABLE V: Internal operation of the quantum network, Fig. 5. A is a single-qubit rotation about the y-axis by $\pi/4$. The gate N flips the qubit.

Q_1^z	Q_2^z	A	CNOT	A	CNOT	$\overline{A}$	CNOT	$\overline{A}$	Desired result
0	0	A		A		$\overline{A}$		$\overline{A}$	$\overline{A}\overline{A}AAQ_3^z = Q_3^z$
0	1	A	N	A		$\overline{A}$	N	$\overline{A}$	$\overline{A}N\overline{A}ANAQ_3^z = Q_3^z$
1	0	A		A	N	$\overline{A}$		$\overline{A}$	$\overline{A}\overline{A}NAAQ_3^z = Q_3^z$
1	1	A	N	A	N	$\overline{A}$	N	$\overline{A}$	$\overline{A}N\overline{A}NANAQ_3^z = NQ_3^z$

Taking into account that V should change only the target qubit (2) if the control qubit (1) is one, we find

$$V_{21}\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-i\pi/4}/\sqrt{2} & 0 & ie^{-i\pi/4}/\sqrt{2} \\ 0 & 0 & 1 & 0 \\ 0 & ie^{-i\pi/4}/\sqrt{2} & 0 & e^{-i\pi/4}/\sqrt{2} \end{pmatrix}\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}. \tag{80}$$

The matrix V_{12} can be brought into a similar form as the controlled phase shift:

$$V_{21} \;=\; \overline{Y}_2 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{-i\pi/2} \end{pmatrix} Y_2 \equiv \overline{Y}_2\overline{R}_{21}(\pi/2)Y_2. \tag{81}$$

The controlled phase shift $R_{21}(\pi/2)$ can be implemented on the Ising model quantum computer (20) by putting $h = -J/2$ and $\tau J = -\pi/2$.

It is easy to see that the quantum network shown in Fig. 4 corresponds to the following sequence of operations:

$$\text{TOFFOLI} = \overline{Y}_3\overline{R}_{31}Y_3\overline{Y}_2I_{21}Y_2\overline{Y}_3R_{32}Y_3\overline{Y}_2I_{21}Y_2\overline{Y}_3\overline{R}_{32}Y_3, \tag{82}$$

where $I_{21} \equiv I_{21}(\pi)$, $R_{ji} \equiv R_{ji}(\pi/2)$, and $\overline{R}_{ji} \equiv R_{ji}(-\pi/2)$. The sequence (82) can be shortened by observing that $\overline{Y}_2I_{21}Y_2\overline{Y}_3 = \overline{Y}_3\overline{Y}_2I_{21}Y_2$. This leads to

$$\text{TOFFOLI} = \overline{Y}_3\overline{R}_{31}\overline{Y}_2I_{21}Y_2R_{32}\overline{Y}_2I_{21}Y_2\overline{R}_{32}Y_3. \tag{83}$$

In Fig. 5, we show a quantum network that performs the same operation as the Toffoli gate up to some known phase factors [9, 57]. The internal operation of this network is summarized in Table V. The first two columns give all possible combinations of the control qubits Q_1^z and Q_2^z. The next seven columns schematically show the individual operations as we move through the network from left to right (see Fig. 5). Entries in the colums labeled "CNOT" having a value "N" indicate that the target qubit (qubit 3) flips. The last column shows the full operation on the third qubit. From Table V, it follows that the first and second row of the last column are trivially satisfied. The third row implies that $N = 1$, which is impossible. Nevertheless, if we choose $A = e^{i\pi S^y/4}$ we find that the condition in the fourth row is satisfied and that $\overline{A}\overline{A}NAA = 2\overline{Y}S^xY = -2S^z$. This means that the target qubit will acquire an extra phase factor (-1) if and only if the first control qubit is one and the second control qubit is zero. Thus, the quantum network of Fig. 5 does not implement a Toffoli gate but performs an operation that is very similar, up to a phase factor that depends on the state of the control qubits [9].

B. Quantum Fourier Transform

The quantum Fourier transform (QFT) is an essential subroutine in, for example, Shor's factoring algorithm, the order finding algorithm and phase estimation in general [9, 36]. The QFT is not a new kind of Fourier transform: it is a particular application of the standard discrete Fourier transform. The latter is defined by $\mathbf{x} = U\mathbf{y}$ where $\mathbf{x} = (x_0, \ldots, x_{N-1})$, $\mathbf{y} = (y_0, \ldots, y_{N-1})$ and $U_{j,k} = e^{2\pi ijk/N}/\sqrt{N}$. Clearly U is a unitary matrix. In the context of quantum computation, the vectors $\mathbf{x}$ and $\mathbf{y}$ are just two different representations of the basis vectors that span the Hilbert space of dimension N. The QFT is defined as

$$\sum_{j=0}^{N-1} a_j|x_j\rangle = \frac{1}{\sqrt{N}} \sum_{j,k=0}^{N-1} e^{2\pi ijk/N} a_j|y_k\rangle = \sum_{k=0}^{N-1} b_k|y_k\rangle, \tag{84}$$

where the amplitudes b_k are the discrete Fourier transform of the amplitudes a_j and $N = 2^L$ where L is the number of qubits. Readers more familiar with traditional quantum mechanics recognize the similarity with the coordinate and momentum representation. Exploiting the massive parallelism of the ideal quantum computer, the QFT can be carried out in $\mathcal{O}(\log_2 N) = \mathcal{O}(L)$ quantum operations [9, 36].

How a quantum network can be derived for the QFT is explained in, for example, Ref. [9, 36]. In Fig. 6, we show a quantum network that performs a four-qubit QFT [36]. The blocks labeled W perform a Walsh-Hadamard transform, and the other blocks perform a controlled phase shift by the angle indicated. Not shown in the network is the series of SWAP-gates that interchange the output qubits (1,4) and (2,3) [9, 36], hence the different labeling of input and output lines in Fig. 6. For the applications that we discuss later, these interchanges merely add to the operation count and can therefore be omitted. To see how the network carries out the QFT (and also for the derivation of the network), the states $|x_j\rangle$ and $|y_j\rangle$ in Eq. (84) have to be written in binary representation, for example, $|x_0\rangle = |0000\rangle$, $|x_1\rangle = |0001\rangle$, and so on.

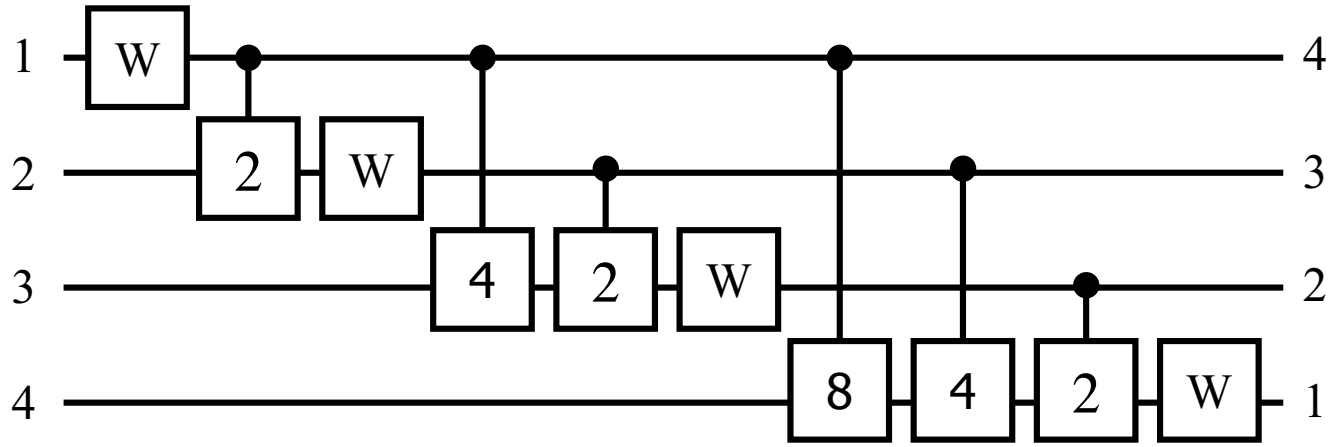

FIG. 6: Quantum network that performs a four-qubit quantum Fourier transform. W denotes the Walsh-Hadamard transform. The operations "2","4", and "8" perform controlled phase shifts with angles $\pi/2$, $\pi/4$, and $\pi/8$, respectively.

C. Finding the Period of a Periodic Function

Assume that we are given the function $f(n) = f(n + M)$ for $n = 0, \ldots, N - 1$. On a quantum computer, we can determine the period M as follows. We use one register of qubits to store n and another one to store $f(n)$. As a first step, the quantum computer is put in the state of uniform superposition over all n. The state of the quantum computer can be written as

$$\frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} |n\rangle|f(n)\rangle \;=\; \frac{1}{\sqrt{N}} \left\{ \sum_{n=0}^{M-1} |n\rangle|f(n)\rangle + \sum_{n=M}^{2M-1} |n\rangle|f(n)\rangle + \ldots \right\}$$

$$= \frac{1}{\sqrt{N}} \sum_{n=0}^{M-1} \left(|n\rangle + |n+M\rangle + \ldots\right) |f(n)\rangle, \tag{85}$$

where, in the last step, we used the periodicity of $f(n)$. Using the Fourier representation of $|n\rangle$ we obtain

$$\begin{aligned} \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} |n\rangle |f(n)\rangle &= \frac{1}{N} \sum_{k=0}^{N-1} \sum_{n=0}^{M-1} e^{2\pi i k n/N} \left(1 + e^{2\pi i k M/N} + e^{4\pi i k M/N} + \ldots + e^{2\pi i k M(L-1)/N}\right) |k\rangle |f(n)\rangle \\ &+ \frac{1}{N} \sum_{k=0}^{N-1} \sum_{n=0}^{L-1} e^{2\pi i k n/N} e^{2\pi i k M L/N} |k\rangle |f(n)\rangle, \end{aligned} \tag{86}$$

where $L = \lfloor N/M \rfloor$ denotes the largest integer L such that $ML \leq N$. In simple terms, L is the number of times the period M fits into the interval $[0, N-1]$. The probability $p_q(M)$ to observe the quantum computer in the state $|q\rangle$ is given by the expectation value of the (projection) operator $Q = |q\rangle\langle q|$. With the restriction on $f(n)$ that $f(n) = f(n')$ implies $n = n'$, we find

$$\langle Q \rangle = p_q(M) = \frac{M}{N^2} \left(\frac{\sin(\pi q M L/N)}{\sin(\pi q M/N)} \right)^2 + \frac{N - ML}{N^2} \frac{\sin(\pi q M(2L+1)/N)}{\sin(\pi q M/N)}. \tag{87}$$

The results for $p_q(M)$ in the case $N = 8$ (3 qubits) are given in Table VI. From Table VI it follows directly that the expectation values of the qubits are ($Q_1^z = Q_2^z = Q_3^z = 0$) if the period $M = 1$, ($Q_1^z = Q_2^z = 0$, $Q_3^z = 0.5$) if the period $M = 2$, ($Q_1^z = 0.5$, $Q_2^z = 0.375$, $Q_3^z = 0.34375$) if the period $M = 3$, and ($Q_1^z = 0$, $Q_2^z = Q_3^z = 0.5$) if the period $M = 4$. Thus, in this simple case the periodicity of $f(n)$ can be unambiguously determined from the expectation values of the individual qubits.

TABLE VI: Probability $p_q(M)$ to observe the state $|q\rangle$ after performing the quantum Fourier transform on the perodic function $f(n) = f(n+M)$ for $n = 0, \ldots, 7$.

q	$p_q(M=1)$	$p_q(M=2)$	$p_q(M=3)$	$p_q(M=4)$
0	1	0.5	0.34375	0.25
1	0	0.0	0.01451	0.00
2	0	0.0	0.06250	0.25
3	0	0.0	0.23549	0.00
4	0	0.5	0.31250	0.25
5	0	0.0	0.23549	0.00
6	0	0.0	0.06250	0.25
7	0	0.0	0.01451	0.00

D. Grover's Database Search Algorithm

Next we consider Grover's database search algorithm to find the needle in a haystack [5, 6]. On a conventional computer, finding an item out of N elements requires $\mathcal{O}(N)$ queries [50]. Grover has shown that a quantum computer can find the item using only $\mathcal{O}(\sqrt{N})$ attempts [5, 6, 100]. Assuming a uniform probability distribution for the needle, for $N = 4$ the average number of queries required by a conventional algorithm is 9/4 [16, 50]. With Grover's quantum algorithm, the correct answer for this database with four items can be found in a single query [14, 16]. Grover's algorithm for the four-item database can be implemented on a two-qubit quantum computer.

The key ingredient of Grover's algorithm is an operation that replaces each amplitude of the basis states in the superposition by two times the average amplitude minus the amplitude itself. This operation is called "inversion about the mean" and amplifies the amplitude of the basis state that represents the searched-for item [5, 6]. To see how this works, it is useful to consider an example. Consider a database containing four items and functions $g_j(x)$, $j = 0, \ldots, 3$ that upon query of the database returns minus one if $x = j$ and plus one if $x \neq j$. Let us assume that

the item to search for corresponds to, for example, 2 ($g_2(0) = g_2(1) = g_2(3) = 1$ and $g_2(2) = -1$). Using the binary representation of integers, the quantum computer is in the state (up to an irrelevant phase factor as usual)

$$|\Phi\rangle = \frac{1}{2}(|00\rangle + |01\rangle - |10\rangle + |11\rangle). \tag{88}$$

The operator B that inverts states like (88) about their means reads

$$B \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}. \tag{89}$$

Applying B to $|\Phi\rangle$ results in $B|\Phi\rangle = |10\rangle$, that is, the correct answer. In general, for more than two qubits, more than one application of B is required to get the correct answer [5, 6]. In this sense the two-qubit case is somewhat special.

Implementation of this example on a two-qubit quantum computer requires a representation in terms of elementary rotations of the preparation and query steps and of the operation of inversion about the mean. Initially, we bring the quantum computer in the state $|00\rangle$ and then transform $|00\rangle$ to the state (88) by a two-step process. First we use the Walsh-Hadamard transform to bring the quantum computer in the uniform superposition state: $W_2W_1|00\rangle = (|00\rangle + |01\rangle + |10\rangle + |11\rangle)/2$, where W_j is given by Eq. (75). Next we apply a transformation F_2 that corresponds to the application of $g_2(x)$ to the uniform superposition state

$$F_2 \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}. \tag{90}$$

This transformation can be implemented by first letting the system evolve in time

$$GW_2W_1|00\rangle = \frac{e^{-i\pi S_1^z S_2^z}}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) = \frac{1}{2}(e^{-i\pi/4}|00\rangle + e^{+i\pi/4}|01\rangle + e^{+i\pi/4}|10\rangle + e^{-i\pi/4}|11\rangle), \tag{91}$$

where the two-qubit operation G is defined by

$$G \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} e^{-i\pi/4} & 0 & 0 & 0 \\ 0 & e^{+i\pi/4} & 0 & 0 \\ 0 & 0 & e^{+i\pi/4} & 0 \\ 0 & 0 & 0 & e^{-i\pi/4} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}. \tag{92}$$

Then we apply a sequence of single-spin rotations to change the four phase factors such that we get the desired state. The two sequences $Y_jX_j\overline{Y}_j$ and $Y_j\overline{X}_j\overline{Y}_j$ operating on qubit j are particularly useful for this purpose since

$$Y_jX_j\overline{Y}_j|0\rangle = e^{+i\pi/4}|0\rangle,\ Y_jX_j\overline{Y}_j|1\rangle = e^{-i\pi/4}|1\rangle,\ Y_j\overline{X}_j\overline{Y}_j|0\rangle = e^{-i\pi/4}|0\rangle,\ Y_j\overline{X}_j\overline{Y}_j|1\rangle = e^{+i\pi/4}|1\rangle. \tag{93}$$

We find

$$\begin{aligned} &Y_1X_1\overline{Y}_1Y_2\overline{X}_2\overline{Y}_2 \left[\frac{1}{2}(e^{-i\pi/4}|00\rangle + e^{+i\pi/4}|01\rangle + e^{+i\pi/4}|10\rangle + e^{-i\pi/4}|11\rangle)\right] \\ &= \frac{1}{2}(e^{-i\pi/4}|00\rangle + e^{-i\pi/4}|01\rangle + e^{+3i\pi/4}|10\rangle + e^{-i\pi/4}|11\rangle) = \frac{e^{-i\pi/4}}{2}(|00\rangle + |01\rangle - |10\rangle + |11\rangle). \end{aligned} \tag{94}$$

Thus we can construct the sequence F_j that transforms the uniform superposition state to the state that corresponds to $g_j(x)$:

$$F_0 = Y_1\overline{X}_1\overline{Y}_1Y_2\overline{X}_2\overline{Y}_2G,\ F_1 = Y_1\overline{X}_1\overline{Y}_1Y_2X_2\overline{Y}_2G,\ F_2 = Y_1X_1\overline{Y}_1Y_2\overline{X}_2\overline{Y}_2G,\ F_3 = Y_1X_1\overline{Y}_1Y_2X_2\overline{Y}_2G. \tag{95}$$

Finally, we need to express the operation of inversion about the mean – that is, the matrix B [see (89)] – by a sequence of elementary operations. It is not difficult to see that B can be written as

$$B = W_1W_2 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} W_1W_2 \equiv W_1W_2PW_1W_2. \tag{96}$$

The same approach that was used to implement $g_2(x)$ also works for P $(= -F_0)$ and yields

$$P = -Y_1\overline{X}_1\overline{Y}_1Y_2\overline{X}_2\overline{Y}_2G. \tag{97}$$

The complete sequence U_j operating on $|00\rangle$ reads

$$U_j = W_1W_2PW_1W_2F_jW_2W_1. \tag{98}$$

Each sequence U_j can be shortened by observing that in some cases a rotation is followed by its inverse. Making use of the alternative representations of the Walsh-Hadamard transform W_i [see Eq. (77)], the sequence for, for instance, $j = 1$ can be written as

$$W_1W_2F_1 = X_1X_1\overline{Y}_1\overline{X}_2\overline{X}_2\overline{Y}_2Y_1\overline{X}_1\overline{Y}_1Y_2X_2\overline{Y}_2G = X_1\overline{Y}_1\overline{X}_2\overline{Y}_2G. \tag{99}$$

The optimized sequences U_j read

$$\begin{aligned}
U_0 &= -X_1\overline{Y}_1X_2\overline{Y}_2GX_1\overline{Y}_1X_2\overline{Y}_2G\overline{X}_2\overline{X}_2\overline{Y}_2\overline{X}_1\overline{X}_1\overline{Y}_1,\\
U_1 &= -X_1\overline{Y}_1X_2\overline{Y}_2GX_1\overline{Y}_1\overline{X}_2\overline{Y}_2G\overline{X}_2\overline{X}_2\overline{Y}_2\overline{X}_1\overline{X}_1\overline{Y}_1,\\
U_2 &= -X_1\overline{Y}_1X_2\overline{Y}_2G\overline{X}_1\overline{Y}_1X_2\overline{Y}_2G\overline{X}_2\overline{X}_2\overline{Y}_2\overline{X}_1\overline{X}_1\overline{Y}_1,\\
U_3 &= -X_1\overline{Y}_1X_2\overline{Y}_2G\overline{X}_1\overline{Y}_1\overline{X}_2\overline{Y}_2G\overline{X}_2\overline{X}_2\overline{Y}_2\overline{X}_1\overline{X}_1\overline{Y}_1.
\end{aligned} \tag{100}$$

Note that the quantum algorithms (100) are by no means unique: various alternative expressions can be written down by using the algebraic properties of the Xs and Ys. Straightforward calculations show that $U_j|0\rangle = |j\rangle$. Hence the application of a sequence (100) to the initial state $|0\rangle$ brings the quantum computer in the state that corresponds to the searched-for item.

E. Finding the Order of a Permutation

We consider the problem of finding the order of a permutation. An experimental realization of this quantum algorithm on a five-qubit NMR quantum computer for the case of a permutation of four items is described in Ref. [24]. The problem is defined as follows: Given a permutation P of the integers $\{0, 1, ..., N-1\}$ and an integer $0 \leq y \leq N-1$, the order $r(y)$ is the smallest integer for which $P^r(y)y = y$. Thus, the purpose of the quantum algorithm is to determine $r(y)$, for a given y and permutation P. The theory in this section closely follows Ref. [24]. We therefore also consider the case $N = 4$, as an example.

The basic idea of the quantum algorithm to find the order of a permutation is to exploit the quantum parallelism to compute $P(y)$, $P^2(y)$, $P^3(y)$, and $P^4(y)$ simultaneously and filter out the power $r(y)$ that yields $P^r(y)y = y$. Denoting $f(n) = P^n$, finding $r(y)$ is the same as finding the period of $f(n)$, a problem that is solved by invoking the QFT (see Section IV C).

First we consider the problem of generating a permutation of $N = 4$ items. We need two qubits to specify one of the four items. Using the binary representation of the integers $\{0, 1, 2, 3\}$, it is easy to see that the CNOT operation C_{21} (the right subscript denoting the control bit) corresponds to the permutation (in cycle notation) $P = (0)(2)(13)$, that interchanges items 1 and 3. Likewise, C_{12} generates the permutation $P = (0)(1)(23)$ and $C_{12}C_{21}C_{12}$ is equivalent to the permutation $P = (0)(3)(12)$. The remaining interchanges of two items can be generated by a combination of CNOT gates and NOT operations. Denoting the NOT operation on the jth qubit by N_j, we find that $C_{21}N_2 = N_2C_{21}$ yields $P = (02)(1)(3)$, $C_{12}N_1 = N_1C_{12}$ yields $P = (2)(3)(01)$, and $C_{12}N_2C_{21}C_{12}$ yields $P = (1)(2)(03)$. Using these elementary interchanges, we can construct any permutation.

The quantum network that carries out the quantum algorithm to find the order of a permutation P of four items is shown in Fig. 7. There is a one-to-one mapping from this network onto the quantum algorithm to find the period of the function $f(n)$ (see Section IV C). The first three qubits hold the integer n; qubits 4 and 5 hold $y = 2y_1 + y_0$. The three Walsh-Hadamard operations change the initial state $|000\rangle|y_1y_0\rangle$ into $|uuu\rangle|y_1y_0\rangle$ where we use the label "u" to denote the uniform superposition. Then, we apply the permutations $P^0y, P^1y, \ldots, P^7y$. In the actual implementation, the sequence of CNOT and NOT operations that implement the permutation P need to be replaced by Toffoli and CNOT gates, respectively, because (the power of) P is applied to the fourth and fifth qubit (representing the integer y), conditional on the state of the first three qubits. Finally, we perform a QFT on the first three qubits and consider the value of the first three qubits Q_1^z, Q_2^z, and Q_3^z. As explained before, in this simple case we can extract the period of the function, and hence the order of P acting on y, from the values of Q_1^z, Q_2^z, and Q_3^z.

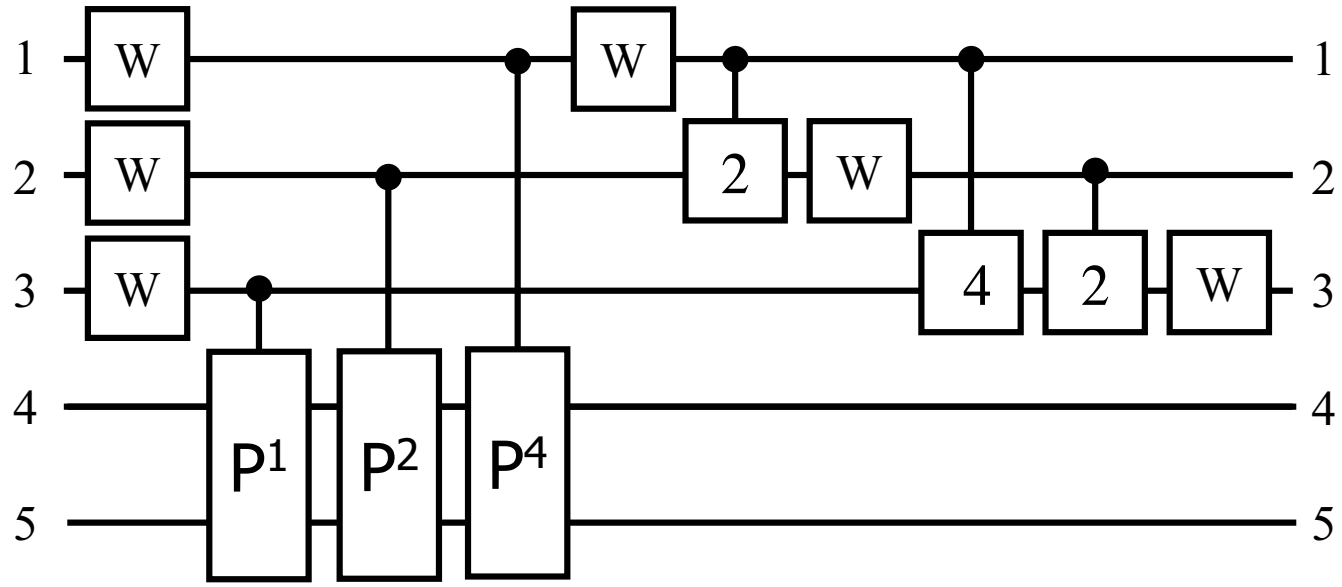

FIG. 7: Quantum network of a quantum algorithm to find the order of a permutation of four items. W and P^k denote the Walsh-Hadamard transform and k applications of the permutation P, respectively. The operations "2" and "4" perform controlled phase shifts with angles $\pi/2$ and $\pi/4$, respectively.

F. Number Factoring: Shor's Algorithm

As another application of the QFT, we consider the problem of factoring integers, that is, Shor's algorithm. For the case $N = 15$, an experimental realization of this quantum algorithm on a seven-qubit NMR quantum computer is given in Ref. [29]. The theory in this section closely follows Ref. [30].

The theory behind Shor's algorithm has been discussed at great length elsewhere [8, 9, 36]. Therefore, we recall only the basic elements of Shor's algorithm. Shor's algorithm is based on the fact that the factors p and q of an integer $N = pq$ can be deduced from the period M of the function $f(j) = a^j \mathrm{mod} N$ for $j = 0, \ldots, 2^n - 1$ where $N \leq 2^n$. Here $a < N$ is a random number that has no common factors with N. Once M has been determined, at least one factor of N can be found by computing the greatest common divisor (g.c.d.) of N and $a^{M/2} \pm 1$.

Compared with the example of the previous section, the new aspect is the modular exponentiation $a^j \mathrm{mod} N$. For $N = 15$ this calculation is almost trivial. Using the binary represention of j, we can write $a^j \mathrm{mod} N = a^{2^{n-1} j_{n-1}} \ldots a^{2 j_1} a^{j_0} \mathrm{mod} N = (a^{2^{n-1} j_{n-1}} \mathrm{mod} N) \ldots (a^{2 j_1} \mathrm{mod} N)(a^{j_0} \mathrm{mod} N) \mathrm{mod} N$, showing that we only need to implement $(a^{2^k j_k} \mathrm{mod} N)$. For $N = 15$ the allowed values for a are $a = 2, 4, 7, 8, 11, 13, 14$. If we pick $a = 2, 7, 8, 13$ then $a^{2^k} \mathrm{mod} N = 1$ for all $k > 1$ while for the remaining cases we have $a^{2^k} \mathrm{mod} N = 1$ for all $k > 0$. Hence for all a we need to "calculate" $1 \mathrm{mod} N$ and $a \mathrm{mod} N$ and for $a = 2, 7, 8, 13$ we in addition have to calculate $a^2 \mathrm{mod} N$. Thus, for $N = 15$, only two (not four) qubits are sufficient to obtain the period of $f(j) = a^j \mathrm{mod} N$ [30]. As a matter of fact, this analysis provides enough information to deduce the factors of $N = 15$ using Shor's procedure so that no further computation is necessary. Nontrivial quantum operations are required if we decide to use three (or more) qubits to determine the period of $f(j) = a^j \mathrm{mod} N$ [30]. Following Ref. [30], we consider a seven-qubit quantum computer with four qubits to hold $f(j)$ and three qubits to perform the QFT.

In essence, the quantum network for the Shor algorithm is the same as the one shown in Fig. 7 (and therefore not shown) with the permutations (two qubits) replaced by modular exponentiation (four qubits). The quantum networks to compute $a^j \mathrm{mod} 15$ for $j = 0, \ldots, 7$ and a fixed input a are easy to construct. For example, consider the case $a = 11 = |1011\rangle$. If j is odd, then $11^j \mathrm{mod} 15 = 11$ and the network should leave $|1011\rangle$ unchanged. Otherwise, $11^j \mathrm{mod} 15 = 1$, and hence it should return $|0001\rangle$ (in this case $M = 2$ and g.c.d.$(N, a^{M/2} \pm 1) =$ {g.c.d.$(15, 10)$, g.c.d.$(15, 12)$} $= \{5, 3\}$, showing that there is no need to perform a quantum computation). The network for this operation consists of two CNOT gates that have as control qubit, the same least-significant qubit of the three qubits that are input to QFT. The sequence of CNOT and Toffoli gates that performs similar operations for the other cases can be found in the same manner.

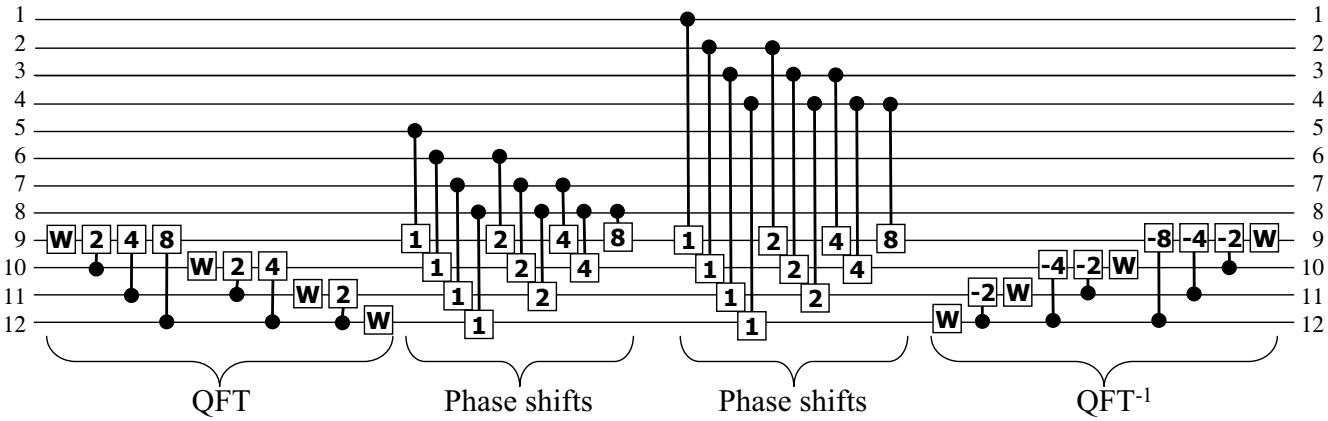

FIG. 8: Quantum network of a three-input adder, as described in Ref. [101]. The algorithm performs a quantum Fourier transform of register 3 (qubits 9 to 12), adds the content of register 2 (qubits 5 to 8) and the content of register 1 (qubits 1 to 4), followed by a QFT on register 3 to yield the final answer (register 1 + register 2 + register 3 mod16). W denotes the Walsh-Hadamard transform. The operations "±1," "±2," "±4," and "±8" perform controlled phase shifts with angles $\pm\pi$, $\pm\pi/2$, $\pm\pi/4$ and $\pm\pi/8$, respectively. Squares that touch each other indicate operations that can be done simultaneously.

G. A Three-Input Adder

This subsection gives another illustration of the use of the QFT: a quantum algorithm to add the content of three four-qubit registers. This example is taken from the Ph.D. thesis of S. Bettelli [101]. The quantum network of the complete circuit is shown in Fig. 8. The modular structure of this approach is clear. Note that with respect to the QFT network of Fig. 6, both the labeling of qubits and the order of operations have been reversed. The former is merely a change of notation and the latter allowed because quantum algorithms are reversible (unitary transformations) by construction [9]. The basis idea of this algorithm is to use the QFT to first transfer the information in a register to the phase factors of the amplitudes and then use controlled phase shifts to add information from the two other registers and finally QFT back to the original representation. Note that this quantum network differs considerably from the one described in Ref. [57] and is also easier to implement.

H. Number Partitioning

As a final example, we discuss a quantum algorithm to count the number of solutions of the number partitioning problem (NPP) [102]. The NPP is defined as follows: Does there exist a partitioning of the set $A = \{a_1, ..., a_n\}$ of n positive integers a_j into two disjoint sets A_1 and $A_2 = A - A_1$ such that the difference of the sum of the elements of A_1 and the sum of the elements of A_2 is either zero (if the sum of all elements of A is even) or one (if the sum of all elements of A is odd)? The following simple example may be useful to understand the problem. If $A = \{1, 2, 3, 4\}$, the answer to the NPP is yes because for $A_1 = \{1, 4\}$ and $A_2 = \{2, 3\}$, the sum of the elements of A_1 and A_2 are both equal to five. In this case there are two solutions because we can interchange A_1 and A_2. If $A = \{1, 1, 1, 4\}$ the answer is again yes, as there is one solution, namely $A_1 = \{1, 1, 1\}$ and $A_2 = \{4\}$. The difference of the sum of the elements of A_1 and A_2 is equal to one, and that is all right because the sum of all elements of A is odd. If $A = \{2, 2, 2, 4\}$, there is no solution to the NPP.

The quantum network for the NPP quantum algorithm for the case that the sum of four integers to be partioned is maximally equal to 16 is shown in Fig. 9. The basic idea behind this quantum algorithm is a number of transformations that reduce the counting of the number of solutions of the NPP to finding the number of zero eigenvalues of a spin-1/2 Hamiltonian [102]. The largest part (in terms of the number of operations) of the quantum algorithm is a combination of gates that transforms the state of the quantum computer such that the number of solutions n_s is a physically measureable quantity (the expectation values of the 15th spin). For $n = 4$, we have $n_s = 16\sqrt{Q_{15}^z}$ [102].

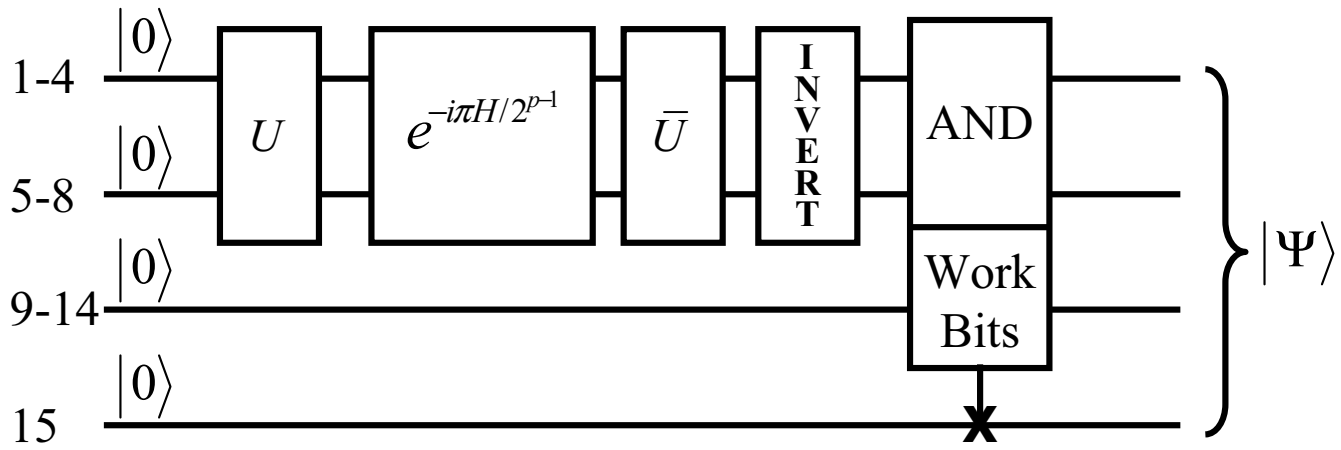

FIG. 9: Block diagram of the quantum algorithm that solves the number partitioning problem. The first four qubits are used to represent the sum of the four integers to be partitioned. Qubits 5 to 8 are used to determine the number of solutions of the number partitioning problem. The remaining seven qubits are used to relate the number of solutions n_s to a physically measurable quantity: the expectation value of the 15th qubit. The unitary transformation U prepares the uniform superposition of the first eight qubits, $\overline{U}$ is the inverse of U, and the combination of INVERT and AND gates sets the 15th qubit to one if and only if the first eight qubits are all one.

V. SIMULATION OF IDEAL QUANTUM COMPUTERS

On the most abstract level, a quantum algorithm that runs on an ideal quantum computer performs one unitary transformation ($D \times D$ matrix) of the state vector (of length $D = 2^L$, where L is the number of qubits). On a conventional computer, this calculation would take $\mathcal{O}(D^2)$ arithmetic operations. As in the case of programming a conventional computer, it is extremely difficult to write down this one-step operation explicitly. Usually an algorithm consists of many steps, each step being a simple construct that easily translates into the elementary operations of the processing units.

As explained in Section II C, for a quantum computer the set of single-qubit rotations and the CNOT gate or any other set of gates that can be used for universal quantum computation will qualify as elementary operations. Thus, a first task is to decompose a given quantum algorithm (unitary matrix) into a sequence of unitary matrices that act on one or two qubits. Recall that each of these matrices is very sparse (see Section III A). Multiplying a very sparse matrix and a vector can be done very efficiently on a conventional computer, typically in $\mathcal{O}(D)$ arithmetic operations. Hence the remaining important question is whether a quantum algorithm can be broken down such that the number of elementary operations is much less than D. The general answer to this question is no [9]. However, for all quantum algorithms reviewed in this chapter, the answer is affirmative. Finding the shortest decomposition of a given unitary matrix in terms of a fixed set of sparse unitary matrices is a difficult optimization problem.

Assuming that we have a representation of a quantum algorithm in terms of the elementary set of sparse unitary matrices, simulating this algorithm on a conventional computer is conceptually simple. The basic techniques are reviewed in Section III A. Software that performs these calculations is called a quantum computer gate-level simulator. It is an essential tool to validate the gate-level design of a quantum algorithm on an ideal quantum computer. Examples of such simulators are discussed in Section VII.

VI. SIMULATION OF PHYSICAL MODELS OF QUANTUM COMPUTERS

The qubits on an ideal quantum computer are ideal two-state quantum systems. Therefore, the operation of an ideal quantum computer does not depend on the intrinsic dynamics of its qubits. However, a physically realizable quantum computer is a many-body system of qubits. The quantum dynamics of these qubits are at the heart of real quantum computation. Conceptually, the main differences between simulating quantum computers on a quantum-gate level (see Section V) and physical models of quantum computers are that the dynamics of the latter are specified in terms of a time-dependent Hamiltonian (not in terms of unitary matrices) and that the physical quantum computer is a single physical system (not a collection of abstract mathematical entities that can be controlled with infinite precision).

There are many proposals for building quantum computer hardware [10, 103–115], and the criteria that a physical system should satisfy to qualify for quantum computation have been examined [116–119]. Candidate technologies for building quantum gates include ion traps, cavity QED, Josephson junctions, and nuclear magnetic resonance (NMR) technology [9–32, 103–115, 120–126]. With the exception of NMR techniques, none of these technologies has demonstrated nontrivial (that is, more than one CNOT operation on an arbitrary linear superposition) quantum computation. In this respect, the field of simulating quantum computers is much more developed: it is relatively easy to simulate realistic physical models of NMR quantum computers [93, 127] or a pair of Josephson junction qubits [128, 129].

In some cases (for instance, electron spins, nuclear spins), there is a one-to-one mapping from the mathematical object of a qubit to a dynamical variable in the quantum mechanical description. In other cases (for example, Josephson junctions), this mapping is not self-evident and requires a very careful analysis of the dynamics of the physical system [128, 129]. Mathematically, nothing prevents us from using three- or even many-level systems as basic units for quantum computation. However, inventing useful quantum algorithms that use two-state systems as qubits already seems to be a major tour de force. Therefore, it is not clear whether working with three- or more-valued qubits instead of two-state systems will bring more than additional complications. Thus, in the following we take the point of view that the quantum computation will be carried out with physical systems that are described by (but not necessarily are) two-state quantum systems.

The physical system defined by Eq. (21) is sufficiently general to serve as a physical model for a generic quantum computer at zero temperature. For instance, Eq. (21) includes the simplest (Ising) model of a universal quantum computer [51, 59]

$$H_{Ising} = -\sum_{i,j=1}^{L} J_{i,j}^{z}(t) S_i^z S_j^z - \sum_{j=1}^{3} \sum_{\alpha=x,y,z} h_j^{\alpha}(t) S_j^{\alpha}. \tag{101}$$

More specific candidate hardware realizations of Eq. (21) include linear arrays of quantum dots [111], Josephson junctions [20], and NMR systems [14–17, 25–28]. An approximate model for the linear arrays of quantum dots reads

$$H(t) = -\sum_{j=1}^{L} E_j S_j^z S_{j+1}^z - \sum_{j=1}^{L} h_j^x(t) S_j^x + E_0 \sum_{j=1}^{L} P_j(t) S_j^z, \tag{102}$$

where $E_j = E_0$ ($E_j = 2E_0$) when j is odd (even) and $h_j^x(t)$ and $P_j(t)$ are external control parameters [111].

Projection of the Josephson-junction model onto a subspace of two states per qubit yields [104, 120]

$$H(t) = -2E_I(t) \sum_{j=1}^{L} S_j^y S_{j+1}^y - E_J \sum_{j=1}^{L} S_j^x - \sum_{j=1}^{L} h_j^z(t) S_j^z, \tag{103}$$

where the energy of the Josephson tunneling is represented by E_J and $E_I(t)$ denotes the energy associated with the inductive coupling between the qubits [104, 120]. Here $h_j^z(t)$ and $E_I(t)$ may be controlled externally.

For nuclei with spin quantum number $S = 1/2$, the Hamiltonian that describes the interacting spin system is of the form (21) [130]. NMR uses radio-frequency electromagnetic pulses to rotate the spins [130, 131]. By tuning the radio frequency of the pulse to the precession (Larmor) frequency of a particular spin, the power of the applied pulse (= intensity times duration) controls how much the spin will rotate. The axis of the rotation is determined by the direction of the applied RF-field. By selecting the appropriate radio-frequency pulses, arbitrary single-spin rotations can be carried out. In other words, using radio-frequency pulses we can perform any single-qubit operation. The simplest model for the interaction of the spins with the external magnetic fields reads

$$h_j^{\alpha}(t) = \hat{h}_j^{\alpha} + \widetilde{h}_j^{\alpha} \sin(2\pi f_j^{\alpha} t + \varphi_j^{\alpha}), \tag{104}$$

where $\hat{h}_j^{\alpha}$ and $\widetilde{h}_j^{\alpha}$ represent the static magnetic field and radio-frequency field acting on the jth spin, respectively. The frequency and phase of the periodic field are denoted by f_j^{α} and φ_j^{α}. The static field $\hat{h}_j^{\alpha}$ fixes the computational basis. It is convention to choose $\hat{h}_j^x = \hat{h}_j^y = 0$ and $\hat{h}_j^z \neq 0$ so that the direction of the static field corresponds to the z-axis of the spins. Under fairly general conditions and to a very good approximation, the nuclear spin Hamiltonian (containing exchange interactions, dipole-dipole interactions, and so on) reduces to the universal quantum computer model (20) in which there are interactions only between the z-components of the spin operators. All NMR quantum computer experiments to date have been interpreted with this model. Therefore, it makes sense to use model (20) as a starting point for simulating physical realizations of quantum computers.

A. NMR-like Quantum Computer

In this section, we illustrate the difference between simulating an ideal, computer science-type quantum computer and a more realistic, physical model of quantum computer hardware [93, 127]. We limit our presentation to the implementation of the CNOT gate and Grover's algorithm on a two-qubit, NMR-like quantum computer. A more extensive discussion, as well as many other examples, can be found in Ref. [127]. All simulations have been carried out using the quantum computer emulator software [132]. The examples given in this section are included in the software distribution of the quantum computer emulator software [132].

As a prototype quantum computer model, we will take the Hamiltonian for the two nuclear spins of the ^{1}H and ^{13}C atoms in a carbon-13 labeled chloroform molecule that has been used in NMR-quantum computer experiments [16, 17]. The strong external magnetic field in the z-direction defines the computational basis. If the spin is aligned along the direction of the field, the state of the qubit is $|0\rangle$; if the spin points in the other direction, the state of the qubit is $|1\rangle$. In the absence of interactions with other degrees of freedom, the Hamiltonian of this spin-1/2 system reads

$$H_{NMR} = -J^z_{1,2} S^z_1 S^z_2 - h^z_1 S^z_1 - h^z_2 S^z_2, \tag{105}$$

where $h^z_1/2\pi \approx 500$MHz, $h^z_2/2\pi \approx 125$MHz, and $J \equiv J^z_{1,2}/2\pi \approx -215$Hz [16]. In our numerical work, we use the rescaled model parameters

$$J = -0.43 \times 10^{-6}, \quad h^z_1 = 1, \quad h^z_2 = 1/4. \tag{106}$$

The ratio $\gamma = h^z_2/h^z_1 = 1/4$ expresses the difference in the gyromagnetic ratio of the nuclear spin of the ^{1}H and ^{13}C atom.

1. Single-Qubit Operations

In NMR experiments, it is impossible to shield a particular spin from the sinusoidal field. An application of a sinusoidal field not only affects the state of the resonant spin but also changes the state of the other spins (unless they are both perfectly aligned along the z-axis). An analytical, quantitative analysis of this simple-looking many-body problem is rather difficult. The values of the model parameters (106) suggest that the interaction between the spins will have a negligible impact on the time evolution of the spins during application of the sinusoidal pulse if the duration of the pulse is much shorter than $1/J$ (we use units such that τJ is dimensionless). Thus, as far as the single-qubit operations are concerned, we may neglect the interaction between the two spins (which is also confirmed by numerical simulation of (105), see later). In this section, we closely follow [127].

We consider the two-spin system described by the time-dependent Schrödinger equation

$$i\frac{\partial}{\partial t}|\Phi(t)\rangle = -\left[h^z_1 S^z_1 + h^z_2 S^z_2 + \widetilde{h}^x_1 (S^x_1 \sin\omega t + S^y_1 \cos\omega t) + \widetilde{h}^x_2 (S^x_2 \sin\omega t + S^y_2 \cos\omega t)\right]|\Phi(t)\rangle, \tag{107}$$

for two interacting spins in a static and a rotating sinusoidal field. As usual, it is convenient to work in a rotating frame [131]. Substituting $|\Phi(t)\rangle = e^{it\omega(S^z_1+S^z_2)}|\Psi(t)\rangle$, we obtain

$$i\frac{\partial}{\partial t}|\Psi(t)\rangle = -\left[(h^z_1-\omega)S^z_1 + (h^z_2-\omega)S^z_2 + \widetilde{h}^x_1 S^y_1 + \widetilde{h}^x_2 S^y_2\right]|\Psi(t)\rangle. \tag{108}$$

Our aim is to determine the conditions under which we can rotate spin 1 by an angle φ_1 without affecting the state of spin 2. First we choose

$$\omega = h^z_1, \tag{109}$$

that is, the frequency of the sinusoidal field is tuned to the resonance frequency of spin 1. Then (108) can easily be integrated. The result is

$$|\Phi(t)\rangle = e^{ith^z_1(S^z_1+S^z_2)} e^{it\widetilde{h}^x_1 S^y_1} e^{it\mathbf{S}_2\cdot\mathbf{v}_{12}}|\Phi(0)\rangle, \tag{110}$$

where $\mathbf{v}_{nm} \equiv (0, \widetilde{h}_m^x, h_m^z - h_n^z)$. The third factor in (110) rotates spin 2 about the vector $\mathbf{v}_{12}$. This factor can be expressed as

$$e^{it\mathbf{S}_m \cdot \mathbf{v}_{nm}} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cos \frac{t|\mathbf{v}_{nm}|}{2} + i|\mathbf{v}_{nm}|^{-1} \begin{pmatrix} h_m^z - h_n^z & -i\widetilde{h}_m^x \\ i\widetilde{h}_m^x & h_n^z - h_m^z \end{pmatrix} \sin \frac{t|\mathbf{v}_{nm}|}{2}, \tag{111}$$

and we see that the sinusoidal field will not change the state of spin 2 if and only if the duration t_1 of the pulse satisfies

$$t_1|\mathbf{v}_{12}| = t_1\sqrt{(h_1^z - h_2^z)^2 + (\widetilde{h}_2^x)^2} = 4\pi n_1, \tag{112}$$

where n_1 is a positive integer. The second factor in (110) is a special case of (111). Putting

$$t_1\widetilde{h}_1^x = \varphi_1, \tag{113}$$

the second factor in (110) will rotate spin 1 by φ_1 about the y-axis. Therefore, if conditions (109), (112), and (113) are satisfied, we can rotate spin 1 by φ_1 without affecting the state of spin 2, independent of the physical realization of the quantum computer. Combining these conditions yields the so-called $2\pi k$ method for suppressing nonresonant effects [109, 133]:

$$t_1 = \frac{\pi}{|h_1^z(1-\gamma)|}\sqrt{16n_1^2 - (\varphi_1/\pi)^2}. \tag{114}$$

If we would use the conditions (109), (113) and (114) to determine the parameters ω, $\widetilde{h}_1^x$ and t_1 of the radio-frequency pulse, the first factor in (110) can still generate a phase shift. This phase shift clearly depends on the state of the spins. Although it drops out in the expression of the expectation value of single qubits, for quantum computation purposes it has to be taken into account (as is confirmed by numerical calculations [127]). Adding the condition

$$t_1 h_1^z = 4\pi k_1, \tag{115}$$

where k_1 is a positive integer ($h_i^z > 0$ by definition), the first factor in (110) is always equal to one. A last constraint on the choice of the pulse parameters comes from the fact that

$$h_2^\alpha = \gamma h_1^\alpha \quad , \quad \widetilde{h}_2^\alpha = \gamma \widetilde{h}_1^\alpha \quad ; \quad \alpha = x, y, z. \tag{116}$$

Without loss of generality, we may assume that $0 < \gamma < 1$.

Using conditions (109), (112), (113), (115), and (116) and reversing the role of spin 1 and spin 2, we obtain

$$(1-\gamma)^2 k_1^2 + \frac{\gamma^2}{4}\left(\frac{\varphi_1}{2\pi}\right)^2 = n_1^2 \quad , \quad (1-\frac{1}{\gamma})^2 k_2^2 + \frac{1}{4\gamma^2}\left(\frac{\varphi_2}{2\pi}\right)^2 = n_2^2, \tag{117}$$

where k_1, k_2, n_1, and n_2 are positive integers. The angles of rotation about the y-axis can be chosen such that $0 \leq \varphi_1 \leq 2\pi$ and $0 \leq \varphi_2 \leq 2\pi$. Of course, similar expressions hold for rotations about the x-axis.

In general, (117) has no solution, but a good approximate solution may be obtained if γ is a rational number and k_1 and k_2 are large. Let $\gamma = N/M$ where N and M are integers satisfying $0 < N < M$. It follows that the representation $k_1 = kMN^2$ and $k_2 = kNM^2$ will generate sufficiently accurate solutions of (117) if the integer k is chosen such that $2kNM(M-N) \gg 1$. In terms of k, N, and M, the relevant physical quantities are then given by

$$\frac{t_1 h_1^z}{2\pi} = 2kMN^2 \quad , \quad \frac{\widetilde{h}_1^x}{h_1^z} = \frac{1}{2kMN^2}\frac{\varphi_1}{2\pi}, \quad , \quad \frac{t_2 h_1^z}{2\pi} = 2kM^3 \quad , \quad \frac{\widetilde{h}_2^x}{h_1^z} = \frac{1}{2kM^3}\frac{\varphi_2}{2\pi}. \tag{118}$$

We have derived conditions (118) under the assumption of ideal sinusoidal RF pulses. In an experiment, there is no such limitation: the sinusoidal fields may be modulated by almost any waveform [130, 134]. However, the fact that

in quantum computer applications it is necessary to use single-spin pulses that do not change the state of the other spins remains. For general pulses, finding the form of the pulse that rotates spin 1 such that the state of spin 2 is not affected is a complicated nonlinear optimization problem [127, 135].

To summarize: If conditions (109), (112), (113), and (115) are satisfied, we can rotate spin 1 by φ_1 without affecting the state of spin 2 and without introducing unwanted phase shifts. In numerical experiments, Eq.(118) may be used to determine the duration of the sinusoidal pulses. These sinusoidal pulses will then be optimized in the sense that a pulse that rotates spin 1 (2) will change the state of spin 2 (1) only slightly if k satisfies $2kNM(M-N) \gg 1$.

2. Two-Qubit Operations

In Section II F, we implemented the CNOT sequence (38) using the Ising model $H = -JS_1^zS_2^z - h(S_1^z + S_2^z)$. The implementation of the CNOT operation using Eq. (105) requires additional steps to account for the fact that the two nuclear spins feel different static fields. The additional rotations are

$$\mathrm{CNOT} = \overline{Y}_2 e^{-i\tau(h_1^z-h)S_1^z} e^{-i\tau(h_2^z-h)S_2^z} e^{-i\tau H_{NMR}} Y_2 = \overline{Y}_2 e^{-i\tau(h_1^z-h)S_1^z} e^{-i\tau(h_2^z-h)S_2^z} Y_2 \overline{Y}_2 e^{-i\tau H_{NMR}} Y_2, \tag{119}$$

where we used the fact that $Y_2\overline{Y}_2 = 1$. The extra phase shifts in (119) can be expressed in terms of single-qubit operations. The identities

$$e^{-i\tau(h_1^z-h)S_1^z} = Y_1X_1'\overline{Y}_1 = \overline{X}_1Y_1'X_1, \quad e^{-i\tau(h_2^z-h)S_2^z} = Y_2X_2'\overline{Y}_2, \tag{120}$$

define the single-spin rotations X_1', Y_1', and X_2'.

In the case of Grover's database search algorithm, the representation of G in terms of the time evolution of (105) reads

$$G = e^{-i\pi S_1S_2} = e^{-i\tau h_1^z S_1^z} e^{-i\tau h_2^z S_2^z} e^{-i\tau H_{NMR}} = Y_2X_2''\overline{Y}_2Y_1X_1''\overline{Y}_1 e^{-i\tau H_{NMR}}, \tag{121}$$

where $\tau = -\pi/J$. This choice of τ also fixes the angles of the rotations and all parameters of the operations X_1'' and X_2''.

Equation (120) suggests that there are many different, logically equivalent sequences that implement the CNOT gate on an NMR-like quantum computer. We have chosen to limit ourselves to the respresentations

$$\mathrm{CNOT}_1 = Y_1X_1'\overline{Y}_1X_2'\overline{Y}_2I'Y_2, \quad \mathrm{CNOT}_2 = Y_1X_1'X_2'\overline{Y}_1\overline{Y}_2I'Y_2, \quad \mathrm{CNOT}_3 = \overline{X}_1Y_1'X_2'\overline{Y}_2X_1I'Y_2, \tag{122}$$

where we introduced the symbol I' to represent the time evolution $e^{-i\tau H_{NMR}}$ with $\tau = -\pi/J$.

On an ideal quantum computer, there is no difference between the logical and physical computer and the sequences (122) give identical results. However, on a physical quantum computer such as the NMR-like quantum computer (105), this is not the case. On a physically realizable NMR-like quantum computer $X_1X_2 \neq X_2X_1$ unless $2kNM(M-N) \gg 1$ and (118) are satisfied *exactly*. Next we use the sequences (122) to demonstrate that this unpleasant feature of physical quantum computers may give rise to large systematic errors.

3. Simulation Results

The model parameters for the rotating sinusoidal fields are determined according to the theory outlined previously. We use the integer k to compute all free parameters and label the results of the quantum computer calculation by the subscript $s = 2kMN^2$. For reference we present the set of parameters corresponding to $s = 8$ ($k = 1$) in Table VII. Multiplying s (the duration of the sinusoidal pulse) with the unit of time (2 ns for the case at hand) shows that in our simulations, single-qubit operations are implemented by using short pulses that are, in NMR terminology, nonselective and hard. In contrast to the analytical treatment given in Section VI A 1, in all our simulations the interaction J is nonzero.

The two-qubit operation I' can be implemented by letting the system evolve in time according to Hamiltonian H_{NMR}, given by (105). I' is the same for both an ideal or NMR-like quantum computer. Note that the condition $\tau J = -\pi$ yields $\tau/2\pi = 1162790.6977$, a fairly large number [compared to $h_1^z = 1$, see (105)]. Also note the digits after the decimal point: this accuracy is necessary for correct operation of the quantum computer [127].

TABLE VII: Model parameters of single-qubit operations on an NMR-like quantum computer using rotating sinusoidal fields for the case ($k = 1$, $N = 1$, $M = 4$); see (118). Parameters of model (107) that do not appear in this table are zero, except for the interaction $J = -0.43 \times 10^{-6}$ and the constant magnetic fields $h_1^z = 1$ and $h_2^z = 0.25$. The time-dependent Schrödinger equation is solved using a time step $\delta/2\pi = 0.01$.

	$\tau/2\pi$	ω	$\widetilde{h}_1^x$	$\widetilde{h}_2^x$	φ_x	$\widetilde{h}_1^y$	$\widetilde{h}_2^y$	φ_y
X_1	8	1.00	-0.0312500	-0.0078125	$-\pi/2$	-0.0312500	-0.0078125	0
X_2	128	0.25	-0.0078125	-0.0019531	$-\pi/2$	-0.0078125	-0.0019531	0
Y_1	8	1.00	0.0312500	0.0078125	0	0.0312500	0.0078125	$\pi/2$
Y_2	128	0.25	0.0078125	0.0019531	0	0.0078125	0.0019531	$\pi/2$
X_1'	8	1.00	0.0559593	0.0139898	$-\pi/2$	0.0559593	0.0139898	0
X_2'	128	0.25	0.0445131	0.0111283	$-\pi/2$	0.0445131	0.0111283	0
Y_1'	8	1.00	-0.0559593	-0.0139898	0	-0.0559593	-0.0139898	$\pi/2$
X_1''	8	1.00	0.0872093	0.0218023	$-\pi/2$	0.0872093	0.0218023	0
X_2''	128	0.25	0.0523256	0.0130914	$-\pi/2$	0.0523256	0.0130914	0

TABLE VIII: Expectation values of the two qubits as obtained by performing a sequence of five CNOT operations on an NMR-like quantum computer. The initial states $|10\rangle$, $|01\rangle$, $|11\rangle$, and $|singlet\rangle = (|01\rangle - |10\rangle)/\sqrt{2}$ have been prepared by starting from the state $|00\rangle$ and performing exact rotations of the spins. The CNOT operations on the singlet state are followed by a $\pi/2$ rotation of spin 1 to yield a nonzero value of qubit 1. The time $s = \tau/2\pi = 2kMN^2$ determines the duration and strength of the sinusoidal pulses through relations (118), see Table VII for the example of the case $s = 8$. The CNOT operation itself was implemented by applying the CNOT sequence given by (122). On an ideal quantum computer, CNOT^4 is the identity operation. For $s = 256$, all results are exact within an error of 0.01.

	Ideal quantum computer		$s = 8$		$s = 16$		$s = 32$		$s = 64$	
Operation	Q_1^z	Q_2^z	Q_1^z	Q_2^z	Q_1^z	Q_2^z	Q_1^z	Q_2^z	Q_1^z	Q_2^z
$(\mathrm{CNOT}_1)^5\|00\rangle$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$(\mathrm{CNOT}_2)^5\|00\rangle$	0.00	0.00	0.24	0.76	0.50	0.26	0.20	0.07	0.06	0.02
$(\mathrm{CNOT}_3)^5\|00\rangle$	0.00	0.00	0.23	0.76	0.50	0.26	0.20	0.07	0.06	0.02
$(\mathrm{CNOT}_1)^5\|01\rangle$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
$(\mathrm{CNOT}_2)^5\|01\rangle$	1.00	1.00	0.76	0.24	0.50	0.74	0.80	0.93	0.95	0.98
$(\mathrm{CNOT}_3)^5\|01\rangle$	1.00	1.00	0.77	0.24	0.50	0.74	0.80	0.93	0.95	0.98
$(\mathrm{CNOT}_1)^5\|10\rangle$	0.00	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.00	1.00
$(\mathrm{CNOT}_2)^5\|10\rangle$	0.00	1.00	0.24	0.24	0.51	0.74	0.20	0.93	0.06	0.98
$(\mathrm{CNOT}_3)^5\|10\rangle$	0.00	1.00	0.23	0.24	0.51	0.74	0.20	0.93	0.06	0.98
$(\mathrm{CNOT}_1)^5\|11\rangle$	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.00
$(\mathrm{CNOT}_2)^5\|11\rangle$	1.00	0.00	0.76	0.76	0.50	0.26	0.80	0.07	0.95	0.02
$(\mathrm{CNOT}_3)^5\|11\rangle$	1.00	0.00	0.77	0.76	0.50	0.26	0.80	0.07	0.95	0.02
$Y_1(\mathrm{CNOT}_1)^5\|singlet\rangle$	1.00	1.00	0.90	1.00	0.03	1.00	0.58	1.00	0.88	1.00
$Y_1(\mathrm{CNOT}_2)^5\|singlet\rangle$	1.00	1.00	0.98	0.24	0.95	0.74	0.98	0.93	0.99	0.98
$Y_1(\mathrm{CNOT}_3)^5\|singlet\rangle$	1.00	1.00	0.79	0.24	0.55	0.74	0.82	0.93	0.95	0.98

As a first check we execute all sequences on an implementation of the ideal quantum computer and confirm that they give the exact answers (results not shown). It is also necessary to rule out that the numerical results depend on the time step δ used to solve the time-dependent Schrödinger equation. The numerical error of the product formula used by QCE is proportional to δ^2 [62]. It goes down by a factor of about one 100 if we reduce the time step by a factor of 10. Within the two-digit accuracy used to present our data, there is no difference between the results for $\delta = 0.01$ and $\delta = 0.001$.

In Table VIII we present simulation results for CNOT^5 acting on one of the basis states and $Y_1\mathrm{CNOT}^5$ acting on a singlet state, using the three logically equivalent but physically different implementations CNOT_1, CNOT_2, and CNOT_3 [see Eq.(122)]. It is clear that some of the least accurate implementations ($s = 8$) do not reproduce the correct answers if the input corresponds to one of the four basis states. Moreover, if the operations act on the exact singlet state, the results strongly depend on the CNOT implementation if $s \leq 32$. In agreement with the theoretical analysis given previously, the exact results are recovered if s is sufficiently large. On the time scale set by J, the pulses used

TABLE IX: Expectation values of the two qubits as obtained by running Grover's database search algorithm on an NMR-like quantum computer. The time $s = \tau/2\pi = 2kMN^2$ determines the duration and strength of the sinusoidal pulses through relations (118), see Table VII for the example of the case $s = 8$. Within two-digit accuracy, all results for $s = 256$ are exact.

	Ideal quantum computer		$s = 8$		$s = 16$		$s = 32$		$s = 64$	
Item position	Q_1^z	Q_2^z	Q_1^z	Q_2^z	Q_1^z	Q_2^z	Q_1^z	Q_2^z	Q_1^z	Q_2^z
0	0.00	0.00	0.48	0.53	0.15	0.16	0.04	0.04	0.01	0.01
1	1.00	0.00	0.52	0.50	0.85	0.15	0.96	0.04	0.99	0.01
2	0.00	1.00	0.55	0.48	0.15	0.84	0.04	0.96	0.01	0.99
3	1.00	1.00	0.45	0.50	0.85	0.85	0.96	0.96	0.99	0.99

in these simulations are so short that the presence of a nonzero J has a negligible effect on the single-qubit pulses. These simulations also demonstrate that in order for a quantum algorithm to work properly, it is not sufficient to show that it correctly operates on the basis states.

In contrast to computation in the classical framework, quantum computation can make use of entangled states. At the point where the quantum algorithm actually uses an entangled state, the quantum algorithm is most sensitive to (accumulated) phase errors. As another illustration of this phenomenon, we present in Table IX some typical results obtained by executing Grover's database search algorithm. We used the same NMR-like quantum computer as for the CNOT calculations. We conclude that reasonably good answers are obtained if $s \geq 32$, in concert with our observations for the CNOT operation.

B. Decoherence

An important topic that we have not discussed so far is the effect of the interaction of the quantum computer with its environment (dissipation, decoherence). The general belief seems to be that engineers will be able to cope with systematic errors (see Section VI A 3 for examples) due to the physics of the qubits and that problems due to decoherence will be the main stumbling block for building useful quantum computers [9]. As most theoretical work [136–153] is based on approximations, the validity of which needs to be established, computer simulation of physical models that include decoherence may be of great value to gain insight into this challenging problem. Dissipation cannot be treated within the context of the simulation models that we have covered in this review. Instead of solving the time-dependent Schrödinger equation, we have to solve the equations of motion of the full density matrix of an interacting many-body system. Although still feasible for a small number of qubits L, the computation time now scales with 2^{2L} instead of with 2^L.

In the absence of dissipation, it is straightforward to incorporate into the class of simulation models physical processes that lead to loss of phase information during the time evolution. In Section III F we have given some hints as how this can be done. Needless to say, performing such simulations is costly in terms of CPU time but the pay-off may be significant: the simple, highly idealized uncorrelated random processes that are being used to analyze error correction and fault-tolerant quantum computing are very far from being physically realistic [35]. Quantum error correction schemes that work well on an ideal quantum computer require many extra qubits and many additional operations to detect and correct errors [35]. The systematic errors discussed previously are not included in the current model [9] of quantum error correction and fault tolerant computing. Our simulation results for a most simple NMR-like quantum computer demonstrate that systematic errors are hard to avoid, in particular if the number of qubits increases (which is a basic requirement for fault-tolerant quantum computation). It would be interesting to simulate a two-qubit quantum computer that interacts with other spins and analyze the effect of decoherence on, for example, the CNOT operation. Such simulations are definitely within reach but, to our knowledge, no results have been reported.

VII. QUANTUM COMPUTER SIMULATORS

A. Review

In this section, we review software tools that run on conventional computers and can be used to study various aspects of quantum computation. The term "quantum computer simulator" is used in a broad sense: not all the software discussed in this section actually simulates a quantum computer. In recent years many quantum computer

simulators have been developed. The level of complexity of the simulations they can perform varies considerably: some deal with both the quantum hardware and software while others focus on quantum computer algorithms and software. An early detailed survey is given in Ref. [154]. In the survey that follows, we confine ourselves to software that was accessible via the Web at the time of writing.

Because quantum computer hardware is not readily available, quantum computer simulators are valuable tools to develop and test quantum algorithms and to simulate physical models for the hardware implementation of a quantum processor. Simulation is an essential part of the design of conventional digital circuits and microprocessors in particular. It is hardly conceivable that it will be possible to construct a real quantum computer without using simulation tools to validate its design and operation.

There is a very important difference in simulating conventional microprocessors and quantum processors. In conventional digital circuits, the internal working of each logical circuit is irrelevant for the logical operation of the processor. However, in a quantum computer the internal quantum dynamics of each elementary building block is a key ingredient of the quantum computer itself. Therefore, in the end the physics of the elementary units that make up the quantum computer have to be incorporated into the simulation model of the quantum computer. It is not sufficient to model a quantum computer in terms of hypothetical qubits that evolve according to the rules of a mathematical model of a hypothetically ideal quantum computer. Including models for noise and errors also is no substitute for the physics of the qubits that comes into play when one wants to make contact to a real physical system.

Quantum computer simulators come in different flavors and have different levels of complexity. A first group of software deals with programming languages for quantum computers. Programming languages express the semantics of a computation in an abstract manner and automatically generate a sequence of elementary operations to control the computer. An overview of quantum programming languages is given in Table X. A second group of simulators comprises the quantum compilers, as summarized in Table XI. A quantum compiler takes as input a unitary transformation and returns a sequence of elementary one-qubit and two-quibit operations that performs the desired quantum operation. Quantum circuit simulators, or gate-level simulators, form the third group of simulators. On an abstract level, quantum computation on an ideal quantum computer amounts to performing unitary transformations on a complex-valued (state) vector. A conventional computer can perform these operations equally well, provided there is enough memory to store all the numbers of the vector. That is exactly what quantum circuit simulators do: they provide a software environment to simulate ideal quantum computers. A summary of quantum circuit simulators is given in Table XII. The fourth group, collected in Table XIII, consists of software that uses time-dependent Hamiltonians to implement the unitary transformations on the qubits. These simulators make it possible to emulate various hardware designs of quantum computers, that is, they simulate models for physical realizations of quantum computers. Simulators of this group can also be used as gate-level simulators. Finally, Table XIV describes a number of purely pedagogical software products.

B. Example: Quantum Computer Emulator

In this section, we briefly discuss some features of QCE (see Table XIII), a software tool that simulates ideal quantum computers and also emulates physical realizations of quantum computers. The QCE is freely distributed as a self-installing executable, containing the program, documentation and many examples of quantum algorithms, including all the quantum algorithms discussed in this review. The file help.htm [155] contains information about how to install and how to start the QCE. The QCE runs in a Windows environment. It consists of a simulator of a generic, general-purpose quantum computer, a graphical user interface, and a real-time visualization module.

QCE's simulation engine is built on the principles reviewed in this chapter. The engine simulates the physical processes that govern the operation of the hardware quantum processor, strictly according to the laws of quantum mechanics. It solves the time-dependent Schrödinger equation (23) by a Suzuki product-formula (see Section III E) in terms of elementary unitary operations. For all practical purposes, the results obtained by this technique are indistinguishable from the exact solution of the time-dependent Schrödinger equation. The graphical user interface is used to control the simulator, to define the hardware of the quantum computer, and to debug and execute quantum algorithms. Using the graphical user interface requires no skills other than the basic ones needed to run a standard MS-Windows applications. The current version of QCE (8.1.1) simulates quantum computers with a maximum of 16 qubits.

QCE can be used to validate designs of physically realizable quantum processors. QCE is also an interactive educational tool to learn about quantum computers and quantum algorithms. As an illustration, we give a short exposition of the implementation of the three-input adder (see Fig. 8) on an ideal quantum computer and of Grover's database search algorithm on an NMR-like quantum computer. Other examples are given in Refs.[93, 102, 127, 156].

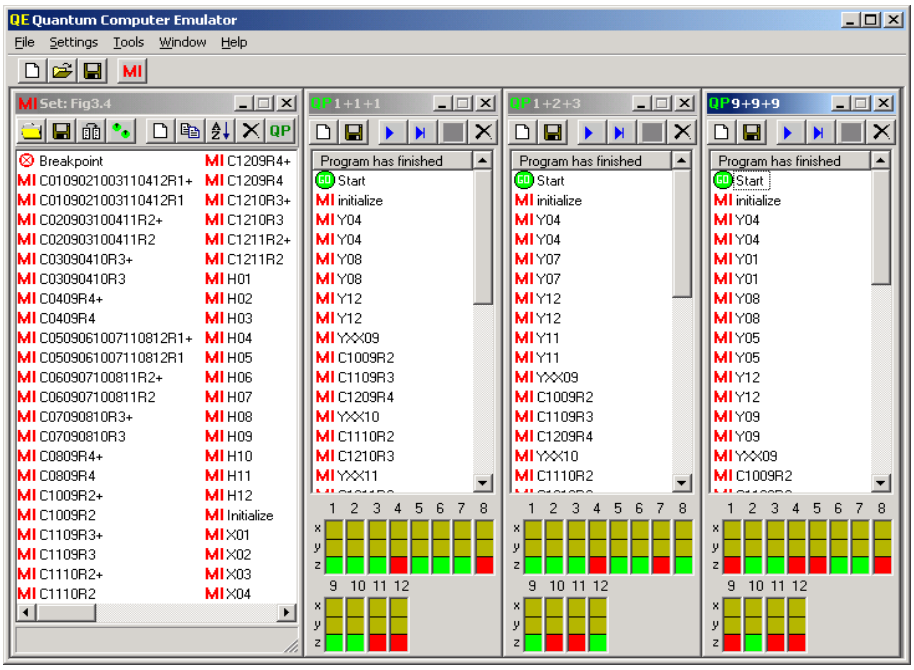

FIG. 10: QCE implementation of the quantum network of Fig. 8 for adding three quantum registers of four qubits each on an ideal 12-qubit quantum computer. Quantum programs (from left to right) compute 1+2+3, 1+1+1, and 9+9+9 mod 16. Not shown are the microinstructions that set the initial values of the three four-qubit registers.

1. *General Aspects*

As described in Section IV, a quantum algorithm for a quantum computer model (21) consists of a sequence of elementary operations that change the state $|\Phi\rangle$ of the quantum processor according to the time-dependent Schrödinger equation (23), namely, by (a product of) unitary transformations. We call these elementary operations microinstructions in the sequel. They do not play exactly the same role as microinstructions in digital processors. They merely represent the smallest units of operation of the quantum processor. The action of a microinstruction on the state $|\Phi\rangle$ of the quantum processor is defined by specifying how long it acts (that is, the time interval it is active) and the values of all the Js and hs appearing in $H(t)$ (21), the model Hamiltonian of the quantum computer. A microinstruction transforms the input state $|\Phi(t)\rangle$ into the output state $|\Phi(t+\tau)\rangle$, where τ denotes the time interval during which the microinstruction is active. During this time interval the only time-dependence of $H(t)$ is through the time-dependence of the external fields on the spins. Microinstructions completely specify the particular (ideal or physical) realization of the quantum computer that one wants to emulate. Quantum algorithms are translated into quantum programs that are written as a sequence of microinstructions. The graphical user interface of the QCE has been developed to facilitate the specification of the microinstructions (to model the quantum computer hardware) and the execution of quantum programs. A detailed exposition of how to use the QCE can be found in Ref. [157].

2. *Three-Input Adder on an Ideal Quantum Computer*

A QCE implementation of the three-input adder described in Section IV G is shown in Fig. 10. The left panel shows the microinstruction set "adder" and the three panels on the right show the quantum programs "1+1+1," "1+2+3," and "9+9+9." These can be found in the quantum program directory "adder." The microinstruction set contains all microinstructions that are needed to execute the quantum programs "1+1+1," "1+2+3," and "9+9+9" on an ideal 12-qubit quantum computer. The results (in binary notation) of the examples (1+1+1, 1+2+3, and 9+9+9) can be read off from the values of qubits 9 to 12 at the bottom of the quantum programs. Qubit 12 corresponds to the least significant bit. The numerical values of the qubits appear when the mouse moves over the bottom region of the quantum program window (red or dark gray corresponds to 1, green or light gray to 0, and greenish brown or middle gray to 0.5). The graphics area in Fig. 10 is too small to show all microinstructions (on the computer the scroll bars allow the user to open/edit all microinstructions). The same holds for the three quantum programs. This example

FIG. 11: Microinstruction set window "grov-rf-k032" together with the quantum program windows "h_cor_1," "grov0," "grov1," and "grov2" that implement Grover's database search algorithm on a two-qubit NMR-like quantum computer for the cases $g_0(x)$, $g_1(x)$, and $g_2(x)$ [see Eq. (100)].

suggests that programming more complicated quantum algorithms like this one should not be done by hand; in fact, the microinstructions and quantum programs for these examples have been generated by another computer program. Some of the microinstructions may look rather complicated, but that is a little misleading: Whenever it is logically allowed to perform operations simultaneously (see Fig. 10), these operations have been put into one microinstruction.

3. Grover's Algorithm on an NMR-like Quantum Computer

The QCE software comes with examples of quantum programs that perform Grover's database search algorithm with four items on ideal and NMR-like quantum computers (105). In this application, the quantum computer has two qubits. In Fig. 11, we show the QCE window after loading the microinstruction set "grov-rf-k032" and the quantum programs "grov0", ..., "grov3" from the QCE program directory "grover2." The quantum programs "grov0", ..., "grov3" implement the Grover algorithms for which the searched-for item corresponds to item 0, 1, 2 or 3, respectively. For printing purposes, quantum program "grov3" has been omitted from Fig. 11. The microinstruction set "grov-rf-k032" corresponds to the parameter set for the rotating sinusoidal fields labeled with $s = 32$ in Table IX. In the microinstruction set "grov-rf-k032," quantum program "h_cor_1" and micro instruction "h_nmr" are used to construct operation G [see (121)]. Running the four quantum programs yields the values for Q_1^z and Q_2^z given in colums eight and nine of Table IX. In contrast to the example of the three-input adder, we use the inverse binary notation: qubit 1 corresponds to the least significant bit.

QCE has an option to visualize the time evolution of the state of the quantum computer in terms of arrows representing the expectation values of the qubits. In Fig. 11 this option was used to visualize the outcome of the program "grov2." Microinstruction sets corresponding to the parameter sets labeled with $s = 8$, $s = 16$, $s = 64$, and $s = 256$ are also provided with the QCE software. Running the programs "grov0", ..., "grov3" with these instruction sets demonstrates that the exact results are recovered if s is sufficiently large (see colums two and three in Table IX).

VIII. SUMMARY AND OUTLOOK

The simulation methods reviewed in this chapter have the potential to faithfully simulate quantum computers of 20 – 30 qubits, considerably more than what is projected to be realizable in the laboratory within the next decade. We say "potential" because whether such simulations can be performed in a reasonable (from the perspective of the researcher or funding organization) amount of real time depends on how the implementation (for example, the vectorization, parallelization, memory usage) efficiently uses conventional computer resources.

An important topic that is virtually unexplored is the implementation of fault-tolerant quantum computation on quantum computers other than the ideal one. The implementation of quantum error correction codes requires additional qubits. It is not known whether fault-tolerant quantum computation is robust with respect to systematic errors that inevitably affect the operation of physically realistic models of quantum computers. It most likely is not, and then it is good to know how to circumvent this problem. The complexity of this problem is such that it is not accessible to theoretical analysis (unless drastic, unrealistic simplifications are introduced), whereas computer simulation may give valuable insight.

Decoherence is another topic that, albeit of much broader scope, is also very important for quantum computation. Although related to the issue of fault-tolerant quantum computation, the effect of the coupling between the quantum computer and its environment on the operation of the quantum computer is a problem that is hardly accessible to theoretical analysis, except perhaps for extremely idealized cases. Also here computer simulation may help.

Finally, we believe that the software approach to quantum computation can help students gain insight into quantum physics during projects in which they improve their computational skills. The algorithms used to simulate quantum computers are fairly generic, that is, they solve differential equations of the parabolic type, and the concepts and techniques learned find applications in many other fields of computational science.

Acknowledgment

We thank V.V. Dobrovitski, A.H. Hams, S. Miyashita, K. De Raedt, and K. Saito for stimulating discussions. Support from the Nederlandse Stichting Nationale Computer Faciliteiten (NCF) is gratefully acknowledged.

TABLE X: Programming languages for quantum computers. URLs last accessed on July 29, 2004.

Name and latest release	Description
QCL (Beta) Version 0.5.1 (0.6.1), March 30, 2004	QCL [158–161] (Quantum Computation Language) is a high-level, architecture-independent programming language for quantum computers, with a syntax derived from classical procedural languages like C or Pascal. Examples such as the quantum Fourier transform, Shor's algorithm, and Grover's algorithm are included. QCL has been developed under Linux; version 0.5.0 compiles with the GNU C++ compiler 3.3. The current version of QCL (sources and i386 Linux binaries) can be downloaded freely from: http://tph.tuwien.ac.at/~oemer/qcl.html
Q language Version 0.5.8, February 18, 2002	Q language [101, 162] is a C++ implementation of a quantum programming language. The source code can be downloaded freely from: http://sra.itc.it/people/serafini/qlang
Quantum Superpositions Version 2.02, April 22, 2003	Quantum Superpositions is a PERL library that enables programmers to use variables that can hold more than one value at the same time. The module can be downloaded freely from: http://search.cpan.org/~lembark/Quantum-Superpositions/lib/Quantum/Superpositions.pm
QuBit, July 24, 2001	QuBit is a C++ library that supports Quantum Superpositions. The library is rewritten starting from the Quantum Superpositions library in PERL. A complete implementation of the QuBit code can be downloaded from: http://www.bluedust.com/qubit/default.asp
Quantum Entanglement Version 0.32, June 5, 2002	Quantum Entanglement is a PERL library that allows the user to put variables in a superposition of states, have them interact with each other, and then observe them. The module can be downloaded freely from: http://search.cpan.org/dist/Quantum-Entanglement
Q-gol Version 3, September 11, 1998	Q-gol is a visual programming language. The fundamental symbols of the system are gates (represented by raised blocks), and directed wires. The graphical editor lets the user select gates, place them on sheets, and wire them together. An implementation of Shor's algorithm is included. The source code can be downloaded freely from: http://www.ifost.org.au/~gregb/q-gol/index.html
Quantum Fog Version 1.6, June 25, 2003	Quantum Fog is a Macintosh application for modeling physical situations that exhibit quantum mechanical behavior. It is a tool for investigating and discussing quantum measurement problems graphically, in terms of quantum Bayesian nets. It simulates a general-purpose quantum computer. The software can be downloaded freely from: http://www.macupdate.com/info.php/id/12181 , http://www.ar-tiste.com
QDD Version 0.2, February 4, 2003	QDD is a C++ library that provides a relatively intuitive set of quantum computing constructs within the context of the C++ programming environment. The emulation of quantum computing is based upon a Binary Decision Diagram representation of the quantum state. Shor's factoring algorithm is included in the QDD library, SHORNUF. The software can be downloaded freely from: http://thegreves.com/david/software
Quantum Lambda Calculus 2003	Functional language based on Scheme for expressing and simulating quantum algorithms. Access at: http://www.het.brown.edu/people/andre/qlambda/index.htmls

TABLE XI: Quantum compilers. URLs last accessed on July 29, 2004.

Name and latest release	Description
Qubiter Version 1.1, April 6, 1999	Qubiter is a quantum compiler written in C++. Qubiter takes as input an arbitrary unitary matrix and returns as output an equivalent sequence of elementary quantum operations. The software can be downloaded freely from: http://www.ar-tiste.com/qubiter.html
GQC, 2002	GQC[163] is an online quantum compiler that returns a circuit for the CNOT in terms of a user-specified unitary transformations. Access at: http://www.physics.uq.edu.au/gqc/

TABLE XII: Quantum circuit simulators. URLs last accessed on July 29, 2004.

Name and latest release	Description
QCAD Version 1.80, May 8, 2003	QCAD is a graphical Windows 98/2000 environment to design quantum circuits. It can export the circuits as BMP or EPS files, simulate the circuits, and show the states of the qubits. QCAD can be downloaded freely from: http://acolyte.t.u-tokyo.ac.jp/~kaityo/qcad
QuaSi(2), March 18, 2002	QuaSi(2) is a general-purpose quantum circuit simulator. It enables the user to build and simulate quantum circuits in a graphical user interface. Demo circuits for Shor's, Grover's and the Deutsch-Josza algorithm are included. QuaSi simulates up to 20 qubits. A Java applet is provided for use over the internet. The full version of QuaSi2 can be downloaded freely from: http://iaks-www.ira.uka.de/QIV/QuaSi/aboutquasi.html
JaQuzzi Version 0.1, January 14, 2001	JaQuzzi [164] is an interactive quantum computer simulator to design, test, and visualize quantum algorithms with up to 20 qubits. The program can either run standalone or as a Web-based applet. To run jaQuzzi, a JAVA virtual machine of version 1.3 or higher is required. The software can be downloaded freely from: http://www.eng.buffalo.edu/~phygons/jaQuzzi/jaQuzzi.html
QCSimulator Version 1.1, March 8, 2000	QCSimulator is a quantum computer simulator for Macintosh and Windows machines. A graphical user interface is used to build a circuit representation of a quantum algorithm and to simulate quantum algorithms by exchanging unitary elements with Mathematica [165]. Shor's factorization algorithm, Grover's database search algorithm, the discrete Fourier transform, and an adder for two numbers are included. The complete software package can be ordered from: http://www.senko-corp.co.jp/qcs/
Libquantum Version 0.2.2, November 3, 2003	Libquantum is a C library for the simulation of an ideal quantum computer. Basic operations for register manipulation such as the Hadamard gate or the Controlled-NOT gate are available. Measurements can be performed on either single qubits or the whole quantum register. Implementations of Shor's factoring algorithm, and Grover's search algorithm are included. Libquantum contains features to study decoherence and quantum error correction. Libquantum is developed on a GNU/Linux platform and requires the installation of a C compiler with complex number support. It can be downloaded freely from: http://www.enyo.de/libquantum
OpenQUACS, May 22, 2000	OpenQUACS [166] (Open-Source QUAntum Computer Simulator) is a library written in Maple that simulates the capabilities of an ideal quantum computer. The simulator comes with a full tutorial. Several quantum algorithms such as Deutsch's algorithm, quantum teleportation, Grover's search algorithm and a quantum adder are included. The software can be downloaded freely from: http://userpages.umbc.edu/~cmccub1/quacs/quacs.html
QuCalc Version 2.13, November 8, 2001	QuCalc is a library of Mathematica functions [165] to simulate quantum circuits and solve problems of quantum computation. The Mathematica package can be downloaded freely from: http://crypto.cs.mcgill.ca/QuCalc
QGAME Version 1, July 7, 2002	QGAME [167, 168] (Quantum Gate And Measurement Emulator) is a system, written in Common Lisp, that allows a user to run quantum computing algorithms on a digital computer. QGAME's graphical user interface (GUI) is a quick hack intended to allow people with no knowledge of Lisp to experiment with QGAME. It uses Macintosh Common Lisp (MCL) interface code and will work only under MacOS with MCL. Not all features of QGAME are available from the GUI. QGAME itself is platform independent (it will run on any platform for which a Common Lisp environment is available); only the GUI requires Macintosh Common Lisp. The full QGAME source code can be downloaded freely from: http://hampshire.edu/lspector/qgame.html
QCompute, July 1997	QCompute is a quantum computer simulator written in Pascal that performs quantum gate operations on an arbitrary number of qubits. The source code and sample gate-networks for a 1-bit and a 2-bit adder can be found in the appendices to a thesis that can be downloaded from: http://w3.physics.uiuc.edu/~menscher/quantum.html
Quantum, July 1997	Quantum is a quantum circuit simulator written in C++ for a Windows environment. It includes an implementation of Shor's algorithm. The software can be downloaded freely from: http://www.themilkyway.com/quantum
Eqcs Version 0.0.5, March 19, 1999	Eqcs is a library allowing clients to simulate a quantum computer. It includes a program showing the creation of a CNOT gate. The software can be downloaded freely from: http://home.snafu.de/pbelkner/eqcs/index.html
QCS, January 11, 2001	QCS (Quantum Computer Simulator) is a quantum computer library in C++. The software can be downloaded freely from: http://www-imai.is.s.u-tokyo.ac.jp/~tokunaga/QCS/simulator.html

TABLE XIII: Quantum Computer Emulators. URLs last accessed on July 29, 2004.

Name and latest release	Description
QCE Version 8.1.1, June 27, 2004	QCE [93, 156] (Quantum Computer Emulator) is a software tool that emulates ideal quantum computers as well as physical implementations of quantum computer hardware. QCE uses time-dependent Hamiltonians and unitary time evolutions to simulate the physical processes that govern the operation of a hardware quantum processor. QCE provides an environment to debug and execute quantum algorithms under realistic experimental conditions. The QCE package includes many examples such as Shor's algorithm for factoring integers, order finding, number partitioning [102], the quantum Fourier transform, various implementations of the Deutsch-Josza algorithm, and Grover's database search on ideal and more realistic quantum computers, such as those used in the 2-qubit NMR quantum computer. The software consists of a Graphical User Interface and the simulator itself. It runs under Windows 98/NT4/2000/ME/(XP with SP1) and Linux+VMware on Intel/AMD processor machines. The software can be downloaded freely from: http://www.compphys.org/qce.htm
QSS, June 2000	QSS [169] (Quantum System Simulator) is a software tool that simulates quantum computations with time-dependent Hamiltonians. It provides a simple and convenient graphical user interface that enables users to specify complex Hamiltonians as sums of smaller ones. The simulator engine is designed as a self-contained module, so that it is independent of the user interface, and can be easily enhanced. The QSS runs on a Windows operating system and can be downloaded freely from: http://web.mit.edu/scottsch/www/qc

TABLE XIV: Pedagogical software. URLs last accessed on July 29, 2004.

Name and latest release	Description
Quantum Turing machine simulator 2002	Mathematica toolkit to construct, run, and research quantum Turing machines. Subscribers to *The Mathematica Journal* can download the toolkit from: http://www.mathematica-journal.com/issue/v8i3/features/hertel
QTM simulator, 1995	Classical simulator of a quantum Turing machine written in C++. The machine has one one-sided infinite tape and one read/write-head and simulates a Fourier transform on a parity function. It can be downloaded freely from: http://www.lri.fr/~durr/Attic/qtm
Quantum Search Simulator, October 10, 2002	Quantum Search Simulator is a Java applet that demonstrates the operation of Grover's quantum search algorithm on a database of four items on a quantum computer based on optical interference. Access at: http://strc.herts.ac.uk/tp/info/qucomp/qucompApplet.html
Grover's algorithm, June 2001	Mathematica-compatible notebook demonstrating Grover's algorithm. It can be downloaded freely from: http://www.cs.caltech.edu/~ chenyang/cs20proj-grover6.nb
CS 596 Quantum Computing, Spring 1999	Matlab programs that demonstrate some features of quantum computing. Demonstrations of the RSA algorithm and Shor's algorithm on a conventional computer are included. It can be downloaded freely from: http://www.sci.sdsu.edu/Faculty/Don.Short/QuantumC/cs662.htm
Shor's algorithm, June 2001	QCL code demonstrating Shor's algorithm. It can be downloaded freely from: http://www.cs.caltech.edu/~ chenyang/myshor.qcl
Shor's algorithm simulator, January 23, 2002	C++ program that simulates the operation of a quantum computer performing Shor's algorithm. It can be downloaded freely from: http://alumni.imsa.edu/~matth/quant

[1] P. Benioff, J. Stat. Phys. **22**, 563 (1980).
[2] R.P. Feynman, Int. J. Theor. Phys. **21**, 467 (1982).
[3] D.R. Simon, *Proceedings of the 35th IEEE Symposium on the Foundations of Computer Science*, edited by S. Goldwasser, IEEE Computer Society, Los Alamitos, CA, 1994, p. 124.
[4] P.W. Shor, *Proceedings of the 35th IEEE Symposium on the Foundations of Computer Science*, edited by S. Goldwasser, IEEE Computer Society, Los Alamitos, CA, 1994, p. 116.
[5] L.K. Grover, in *Proc. 28th Annual ACM Symposium of Theory of Computing*, ACM, Philadelphia, 1996.
[6] L.K. Grover, Phys. Rev. Lett. **79**, 325 (1997).
[7] D.R. Simon, SIAM J. Comput. **26**, 1474 (1997).
[8] P.W. Shor, SIAM Review **41**, 303 (1999).
[9] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, 2000.
[10] J.J. Cirac and P. Zoller, Phys. Rev. Lett. **74**, 4091 (1995).
[11] C. Monroe, D.M. Meekhof, B.E. King, W.M. Itano, and D.J. Wineland, Phys. Rev. Lett. **75**, 4714 (1995).
[12] T. Sleator and H. Weinfurther, Phys. Rev. Lett. **74**, 4087 (1995).
[13] P. Domokos, J.M. Raimond, M. Brune, and S. Haroche, Phys. Rev. A**52**, 3554 (1995).
[14] J.A. Jones and M. Mosca, J. Chem. Phys. **109**, 1648 (1998).
[15] J.A. Jones, M. Mosca, and R.H. Hansen, Nature **393**, 344 (1998).
[16] I.L. Chuang, L.M.K. Vandersypen, X. Zhou, D.W. Leung, and S. Lloyd, Nature **393**, 143 (1998).
[17] I.L. Chuang, N. Gershenfeld, and M. Kubinec, Phys. Rev. Lett. **80**, 3408 (1998).
[18] R.J. Hughes, D.F.V. James, J.J. Gomez, M.S. Gulley, M.H. Holzscheiter, P.G. Kwiat, S.K. Lamoreaux, C.G. Peterson, V.D. Sandberg, M.M. Schauer, C.M. Simmons, C.E. Thorburn, D. Tupa, P.Z. Wang, and A.G. White, Fortschr. Phys. **46**, 329 (1998).
[19] D.J. Wineland, C. Monroe, W.M. Itano, B.E. King, D. Leibfried, D.M. Meekhof, C. Myatt, and C. Wood, Fortschr. Phys. **46**, 363 (1998).
[20] Y. Nakamura, Yu. A. Pashkin, and J.S. Tsai, Nature **398**, 786 (1999).
[21] G. Nogues, A. Rauschenbeutel, S. Osnaghi, M. Brune, J.M. Raimond, and S. Haroche, Nature **400**, 239 (1999).
[22] A. Blias and A. Zagoskin, Phys. Rev. A**61**, 042308 (2000).
[23] M.C. de Oliveira, and W.J. Munro, Phys. Rev. A**61**, 042309 (2000).
[24] L.M.K. Vandersypen, M. Steffen, G. Breyta, C.S. Yannoni, R. Cleve, and I.L. Chuang, Phys. Rev. Lett. **85**, 5452 (2000).
[25] R. Marx, A.F. Fahmy, J.M. Meyers, W. Bernel, and S.J. Glaser, Phys. Rev. A**62**, 012310 (2000).
[26] E. Knill, R. Laflamme, R. Martinez, and C.-H. Tseng, Nature **404**, 368 (2000).
[27] D.G. Cory, R. Laflamme, E. Knill, L. Viola, T.F. Havel, N. Boulant, G. Boutis, E. Fortunato, S. Lloyd, R. Martinez, C. Negrevergne, M. Pravia, Y. Sharf, G. Teklemariam, Y.S. Weinstein, and W.H. Zurek, Fortschr. Phys. **48**, 875 (2000).
[28] J.A. Jones, Fortschr. Phys. **48**, 909 (2000).
[29] L.M.K. Vandersypen, M. Steffen, G. Breyta, C.S. Yannoni, M.H. Sherwood, and I.L. Chuang, Nature **414**, 883 (2001).
[30] L.M.K. Vandersypen, Ph.D. Thesis, Dept. of Electrical Engineering, Stanford University (2001), http://arxiv.org/abs/quant-ph/0205193 (URL last accessed on July 29, 2004).
[31] J.A. Jones, Prog. Nucl. Magn. Spec. **38**, 325 (2001).
[32] S. Takeuchi, Electronics and Communications in Japan **84**, 52 (2001).
[33] C. Zalka, Proc. R. Soc. Lond. A**454**, 313 (1998).
[34] D.P. Landau and K. Binder, *A Guide to Monte Carlo Simulation in Statistical Physics*, Cambridge University Press, Cambridge, 2000.
[35] J. Preskill, Proc. R. Soc. Lond. A**454**, 469 (1998).
[36] A. Ekert and R. Jozsa, Rev. Mod. Phys. **68**, 733 (1996).
[37] C.H. Bennet, E. Bernstein, G. Brassard, and U. Vazirani, SIAM J. Comput. **26**, 1510 (1997).
[38] V. Vedral and M. Plenio, Progress in Quantum Electronics **22**, 1 (1998).
[39] H.E. Brandt, Progr. Quant. Electr. **22**, 257 (1998).
[40] A. Ekert, P. Hayden, H. Inamori, and D. Kuan Li Oi, Int. J. Mod. Phys. A**20**, 3335 (2001).
[41] P.E. Black, D.R. Kuhn, and C.J. Williams, Adv. Comp. **56**, 190 (2002).
[42] A. Galindo and M.A. Martin-Delgado, Rev. Mod. Phys. **74**, 347 (2002).
[43] G.P. Berman, G.D. Doolen, R. Mainieri, and V.I. Tsifrinovich, *Introduction to Quantum Computers*, World Scientific, Singapore, 1998.
[44] H.-K. Lo, S. Popescu, and T. Spiller, *Introduction to Quantum Computation and Quantum Information*, World Scientific, Singapore (1998).
[45] C. Macciavello, G.M. Palma, and Z. Zeilinger, *Quantum Computation and Quantum Information Theory*, World Scientific (2000).
[46] L.I. Schiff, *Quantum Mechanics*, McGraw-Hill, New York (1968).
[47] G. Baym, *Lectures on Quantum Mechanics*, W.A. Benjamin, Reading MA (1974).
[48] L.E. Ballentine, *Quantum Mechanics: A Modern Development*, World Scientific, Singapore (2003).
[49] H. De Raedt, K. Michielsen, S. Miyashita, and K. Saito, Prog. Theor. Phys. Suppl. 145, 233 (2002).

[50] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, 1994.
[51] S. Lloyd, Science **261**, 1569 (1993).
[52] D.P. DiVincenzo, Science **270**, 255 (1995).
[53] D.P. DiVincenzo, Phys. Rev. A**51**, 1015 (1995).
[54] J.M. Myers, Phys. Rev. Lett. **78**, 1823 (1997).
[55] J.H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.
[56] G.H. Golub and C.F. Van Loan, *Matrix Computations*, John Hopkins University Press, Baltimore, 1996.
[57] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J.A. Smolin, and H. Weinfurter, Phys. Rev. A**52**, 3457, 1995.
[58] R. Bellman, *Introduction to Matrix Analysis*, SIAM, Philadelphia, 1997.
[59] G.P. Berman, G.D. Doolen, D.D. Holm, and V.I. Tsifrinovich, Phys. Lett. A**193**, 444 (1994).
[60] M. Suzuki, Proc. Japan Acad. **69** Ser. B, 161 (1993).
[61] G.D. Smith, *Numerical solution of partial differential equations*, Clarendon Press, Oxford, 1985.
[62] H. De Raedt, Comp. Phys. Rep. **7**, 1 (1987).
[63] B.N. Parlett, *The Symmetric Eigenvalue Problem*, Classics in Applied Mathematics, 20, Society for Industrial and Applied Mathematics, (SIAM), Philadelphia, PA, 1998.
[64] H. Tal-Ezer and R. Kosloff, J. Chem. Phys. **81**, 3967, 1984.
[65] H. Tal-Ezer, J. Scient. Comp. **4**, 25, 1989.
[66] C. Leforestier, R.H. Bisseling, C. Cerjan, M.D. Feit, R. Friesner, A. Guldberg, A. Hammerich, G. Jolicard, W. Karrlein, H.-D. Meyer, N. Lipkin, O. Roncero, and R. Kosloff, J. Comp. Phys. **94**, 59, 1991.
[67] T. Iitaka, S. Nomura, H. Hirayama, X. Zhao, Y. Aoyagi, and T. Sugano, Phys. Rev. E**56**, 1222, 1997.
[68] R.N. Silver and H. Röder, Phys. Rev. E**56**, 4822, 1997.
[69] Y.L. Loh, S.N. Taraskin, and S.R. Elliot, Phys. Rev. Lett. **84**, 2290, 2000.
[70] V.V. Dobrovitski and H.A. De Raedt, Phys. Rev. E**67**, 056702, 2003.
[71] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions*, Dover, New York, 1964.
[72] C. Moler and C.F. Van Loan, SIAM Review **20**, 801, 1978.
[73] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes*, Cambridge, New York, 1986.
[74] T.J. Park and J.C. Light, J. Chem. Phys. **85**, 5870 (1986).
[75] U. Manthe, H. Köppel, and L.S. Cederbaum, J. Chem. Phys. **95**, 1708 (1991).
[76] J. Jacklič and P. Prelovšek, Phys. Rev. B.**49**, 5065 (1994).
[77] J. Jacklič and P. Prelovšek, Adv. Phys. **49**, 1 (2000).
[78] H.F. Trotter, Proc. Am. Math. Soc. **10**, 545 (1959).
[79] M. Suzuki, S. Miyashita, and A. Kuroda, Prog. Theor. Phys. **58**, 1377 (1977).
[80] M. Suzuki, J. Math. Phys. **26**, 601 (1985).
[81] H. De Raedt and B. De Raedt, Phys. Rev. A**28**, 3575 (1983).
[82] M. Suzuki, J. Math. Phys. **32**, 400 (1991).
[83] M.D. Feit, J.A. Fleck, and A. Steiger, J. Comput. Phys **47**, 412 (1982).
[84] H. De Raedt and P. de Vries, Z. Phys. B**77**, 243 (1989).
[85] M. Krech, A. Bunker, and D.P. Landau, Comp. Phys. Comm. **111**, 1 (1998).
[86] H. Kobayashi, N. Hatano, and M. Suzuki, Physica A**211**, 234 (1994).
[87] A. Rouhi and J. Wright, Comp. in Phys. **9**, 554 (1995).
[88] T. Kawarabayashi and T. Ohtsuki, Phys. Rev. B**53**, 6975 (1996).
[89] T. Ohtsuki and T. Kawarabayashi, J. Phys. Soc. Jap. **66**, 314 (1997).
[90] B.A. Shadwick and W.F. Buell, Phys. Rev. Lett. **79**, 5189 (1997).
[91] P. Tran, Phys. Rev. E**58**, 8049 (1998).
[92] K. Michielsen, H. De Raedt, J. Przeslawski, and N. García, Phys. Rep. **304**, 89 (1998).
[93] H. De Raedt, A.H. Hams, K. Michielsen, and K. De Raedt, Comp. Phys. Comm. **132**, 1 (2000).
[94] J.S. Kole, M.T. Figge, and H. De Raedt, Phys. Rev. E**64**, 066705 (2001).
[95] P. de Vries and H. De Raedt, Phys. Rev. B**47**, 7929 (1993).
[96] V.V. Dobrovitski, H.A. De Raedt, M.I. Katsnelson, and B.N. Harmon, Phys. Rev. Lett. **90**, 210401 (2003).
[97] V. Vedral, A. Barenco, and A. Ekert, Phys. Rev. A**54**, 147 (1996).
[98] R. Cleve, A. Ekert, C. Machiavello, and M. Mosca, Proc. R. Soc. Lond. A**454**, 339 (1998).
[99] D. Beckman, A.N. Chari, S. Devabhaktuni, and J. Preskill, Phys. Rev. A**54**, 1034 (1996).
[100] C. Zalka Phys. Rev. A**62**, 052305 (2000).
[101] S. Bettelli, Ph.D. Thesis, University of Trento (2002)
[102] H. De Raedt, K. Michielsen, K. De Raedt, and S. Miyashita, Phys. Lett. A**200**, 227 (2001).
[103] B.E. Kane, Nature **393**, 133 (1998).
[104] Y. Makhlin, G. Schön, and A. Shnirman, Nature **398**, 305 (1999).
[105] K. Mølmer, and A. Sørensen, Phys. Rev. Lett. **82**, 1835 (1999).
[106] A. Sørensen, and K. Mølmer, Phys. Rev. Lett. **82**, 1971 (1999).
[107] T.P. Orlando, J.E. Mooij, L. Tian, C.H. van der Wal, L.S. Levitov, S. Lloyd, and J.J. Mazo, Phys. Rev. B**60**, 15398 (1999).
[108] D.P. DiVincenzo, D. Bacon, J. Kempe, G. Burkard, and K.B. Whaley, Nature **408**, 339 (2000).
[109] G.P. Berman, G.D. Doolen, G.V. López, and V.I. Tsifrinovich, Phys. Rev. A**61**, 042307 (2000).

[110] Y. Makhlin, G. Schön, and A. Shnirman, J. Low Temp. Phys. **118**, 751 (2000).
[111] G. Toth, and S. Lent, Phys. Rev. A**63**, 052315 (2001).
[112] D.K. Ferry, R. Akis, and J. Harris, SuperLat. and MicroStruc. **30**, 90 (2001).
[113] J. Harris, R. Akis, and D.K. Ferry, Appl. Phys. Lett. **79**, 2214 (2001).
[114] X. Hu and S. Das Sarma, Phys. Stat. Sol.(b) **238**, 360 (2003).
[115] A. Imamoglu, Physica E **16**, 47 (2003).
[116] D.P. DiVincenzo, Fortschr. Phys. **48**, 9 (2000).
[117] M.I. Dyakonov, Opt. and Spectr. **95**, 261 (2003).
[118] R.W. Keyes, Appl. Phys. A**76**, 737 (2003).
[119] S.D. Bartlett and B.C. Sanders, J. Mod. Opt. **50**, 2331 (2003).
[120] R. Fazio, G.M. Palma, and J. Siewert, Phys. Rev. Lett. **83**, 5385 (1999).
[121] G.P. Berman, G.D. Doolen, and V.I. Tsifrinovich, SuperLat. and MicroStruc. **27**, 90 (2000).
[122] G. Schön, Y. Makhlin, and A. Shnirman, Physica C **352**, 113 (2001).
[123] D.M. Lucas, C.J.S. Donald, J.P. Home, M.J. McDonnell, A. Ramos, D.N. Stacey, J.-P. Stacey, A.M. Steane, and S.C. Webster, Phil. Trans. R. Soc. Lond. A**361**, 1401 (2003).
[124] R.G. Clark, R. Brenner, T.M. Buehler, V. Chan, N.J. Curson, A.S. Dzurak, E. Gauja, H.S. Goan, A.D. Greentree, T. Hallam, A.R. Hamilton, L.C.L. Hollenberg, D.N. Jamieson, J.C. McCallum, G.J. Milburn, J.L. O'Brien, L. Oberbeck, C.I. Pakes, S.D. Prawer, D.J. Reilly, F.J. Ruess, S.R. Schofield, M.Y. Simmons, F.E. Stanley, R.P. Starrett, C. Wellard, and C. Yang, Phil. Trans. R. Soc. Lond. A**361**, 1451 (2003).
[125] A. Ardavan, M. Austwick, S.C. Benjamin, G.A.D. Briggs, T.J.S. Dennis, A. Ferguson, D.G. Hasko, M. Kanai, A.N. Khlobystov, B.W. Lovett, G.W. Morley, R.A. Oliver, D.G. Pettifor, K. Porfyrakis, J.H. Reina, J.H. Rice, J.D. Smith, R.A. Taylor, D.A. Williams, C. Adelmann, H. Mariette, and R.J. Hamers, Phil. Trans. R. Soc. Lond. A**361**, 1473 (2003).
[126] C.H.W. Barnes, Phil. Trans. R. Soc. Lond. A**361**, 1487 (2003).
[127] H. De Raedt, K. Michielsen, A. Hams, S. Miyashita, and K. Saito, Eur. Phys. J. **B27**, 15 (2002).
[128] P.R. Johnson, F.W. Strauch, A.J. Dragt, R.C. Ramos, C.J. Lobb, J.R. Anderson, and F.C. Wellstood, Phys. Rev. B**67**, 020509 (2003).
[129] F.W. Strauch, P.R. Johnson, A.J. Dragt, C.J. Lobb, J.R. Anderson, and F.C. Wellstood, Phys. Rev. Lett. **91**, 167005 (2003).
[130] R. Ernst, G. Bodenhausen, and A. Wokaun, *Principles of Nuclear Magnetic Resonance in One and Two Dimensions*, Oxford University Press, New York (1987).
[131] C.P. Slichter, *Principles of Magnetic Resonance*, Springer, Berlin, 1990.
[132] QCE can be downloaded from `http:/www.compphys.org/qce.htm` (URL last accessed on July 29, 2004).
[133] G.P. Berman, D.K. Campbell, and V.I. Tsifrinovich, Phys. Rev. B**55**, 5929 (1997),
[134] R. Freeman, *Spin Choreography*, Spektrum, Oxford, 1997.
[135] J.P. Palao and R. Kosloff, Phys. Rev. Lett. **89**, 188301 (2002).
[136] L.M. Duan and G.C. Guo, Phys. Rev. A**57**, 737 (1998).
[137] P. Zanardi, Phys. Rev. A**57**, 3276 (1998).
[138] C.P. Sun, H. Zhan, and X.F. Liu, Phys. Rev. A**58**, 1810 (1998).
[139] J.P. Barnes and W.S. Warren, Phys. Rev. A**60**, 1810 (1999).
[140] A. Beige, D. Braun, B. Tregenna, and P. Knight, Phys. Rev. Lett. **85**, 1762 (2000).
[141] A.Y. Smirnov, Phys. Rev. A**67**, 155104 (2003).
[142] C.H. Tseng, S. Somarro, Y. Sharf, E. Knill, R. Laflamme, T.F. Havel, and D.G. Cory, Phys. Rev. A**62**, 032309 (2000).
[143] A.R.R. Carvahlo, P.Milman, R.L. de Matos Filho, and L. Davidovich, Phys. Rev. Lett. **85**, 1762 (2000).
[144] S. Mancini and R. Bonifacio, Phys. Rev. A**63**, 012309 (2001).
[145] M. Thorwart and P. Hänggi, Phys. Rev. A**65**, 012309 (2001).
[146] L.-A. Wu and D.A. Lidar, Phys. Rev. Lett. **88**, 207902 (2002).
[147] S.D. Barret and G.J. Milburn, Phys. Rev. B **68**, 155307 (2003).
[148] B.J. Dalton, J. Mod. Opt. **50**, 951 (2003).
[149] T. Yu and J.H. Eberly, Phys. Rev. B **68**, 165322 (2003).
[150] M.J. Storcz and F.K. Wilhelm, Phys. Rev. A**67**, 042319 (2003).
[151] V. Protopopescu, R. Perez, C. D'Helon, and J. Schmulen, J. Phys. A: Math. Gen **36**, 2175 (2003).
[152] S.-B. Li and J.-B. Xu, Phys. Lett. A**311**, 313 (2003).
[153] G. Teklemariam, E.M. Fortunato, C.C. López, J. Emerson, J.J. Paz, T.F. Havel, and D.G. Cory, Phys. Rev. A**67**, 062316 (2003).
[154] J. Wallace, International Journal of Computing Anticipatory Systems **10**, 230 (2001).
[155] http://rugth30.phys.rug.nl/compphys0/QCE/help.htm (URL last accessed on July 29, 2004).
[156] K. Michielsen, H. De Raedt, and K. De Raedt, Nanotechnology **13**, 23 (2002).
[157] K.F.L. Michielsen and H. De Raedt, Turk. J. Phys. **27**, 343 (2003).
[158] B. Ömer, Master thesis, Technical University of Vienna, 1998, http://tph.tuwien.ac.at/~oemer/qcl.html (URL last accessed on July 29, 2004).
[159] B. Ömer, Master thesis, Technical University of Vienna, 2000, http://tph.tuwien.ac.at/~oemer/qcl.html (URL last accessed on July 29, 2004).
[160] B. Ömer, Ph.D. thesis, Technical University of Vienna, 2003, http://tph.tuwien.ac.at/~oemer/qcl.html (URL last

accessed on July 29, 2004).
[161] B. Ömer, http://arxiv.org/abs/quant-ph/0211100, (URL last accessed on July 29, 2004).
[162] S. Bettelli, T. Calarco, and L. Serafini, Eur. Phys. J. D**25**, 181 (2003).
[163] M.J. Bremner, C.M. Dawson, J.L. Dodd, A. Gilchrist, A.W. Harrow, D. Mortimer, M.A. Nielsen, and T.J. Osborne, Phys. Rev. Lett. **89**, 247902 (2002).
[164] F. Schürmann, Master thesis, State University of New York at Buffalo, 2000, http://www.eng.buffalo.edu/~phygons/jaQuzzi/thesis.html (URL last accessed on July 29, 2004).
[165] *Mathematica* is a registered trademark of Wolfram Research, Inc., http://www.wolfram.com (URL last accessed on July 29, 2004).
[166] C.B. McCubbin, Master thesis, 2000, University of Maryland, http://userpages.umbc.edu/~cmccub1/quacs/quacs.html (URL last accessed on July 29, 2004).
[167] L. Spector, H. Barnum, and H.J. Bernstein, in *Advances in Genetic Programming* **3**, edited by L. Spector, W.B. Langdon, U.-M. OReilley, and P.J. Angeline, Vol.3 p. 135, MIT Press, Cambridge, MA, 1999.
[168] L. Spector, H. Barnum, H.J. Bernstein, and N. Swamy, in *Proceedings of the 1999 Congress Evolutionary Computation*, IEEE Press, Piscatawasy, N.J., 1999, p. 2239.
[169] S. Schneider, Master Thesis, Massachusetts Institute of Technology, 2000, http://web.mit.edu/scottsch/www/qc/ (URL last accessed on July 29, 2004).