Quantum Computing

Peter W. Shor

Abstract. The Church-Turing thesis says that a digital computer is a universal computational device; that is, it is able to simulate any physically realizable computational device. It has generally been believed that this simulation can be made efficient so that it entails at most a polynomial increase in computation time. This may not be true if quantum mechanics is taken into consideration. A quantum computer is a hypothetical machine based on quantum mechanics. We explain quantum computing, and give an algorithm for prime factorization on a quantum computer that runs asymptotically much faster than the best known algorithm on a digital computer. It is not clear whether it will ever be possible to build large-scale quantum computers. One of the main difficulties is in manipulating coherent quantum states without introducing errors or losing coherence. We discuss quantum error-correcting codes and fault-tolerant quantum computing, which can guarantee highly reliable quantum computation, given only moderately reliable quantum computing hardware.

1991 Mathematics Subject Classification: Primary 68Q05; Secondary 11Y05, 81P99.
Keywords and Phrases: Quantum computer, computational complexity, prime factorization.

1 Introduction.

Quantum computers are hypothetical machines that use principles of quantum mechanics for their basic operations. They will be very difficult to build; currently experimental physicists are working on two- and three-bit quantum computers, and useful quantum computers would require hundreds to thousands of bits. However, there seem to be no fundamental physical laws that would preclude their construction. In 1994, I showed that a quantum computer could factor large numbers in time polynomial in the length of the numbers, a nearly exponential speed-up over classical algorithms. This factoring result was surprising for a number of different reasons. First, the connection of quantum mechanics with number theory was itself surprising. For cryptographers, the result was surprising because the difficulty of factoring is the basis of the RSA cryptosystem [27], and nobody had anticipated the possibility of an attack via quantum physics. For many theoretical computer scientists, it was surprising because they had more or less convinced themselves that no type of computing machine could offer this large a speed-up over a classical digital computer. In retrospect, several results [7, 30] should have led them to question this; however, not much attention was paid to these results until they led to the development of the factoring algorithm,

A question that has generated much discussion is where the extra power of quantum computers comes from. There are a number of differences between quantum and classical computers, and most appear to be required for the extra power. In particular, quantum interference is needed; one high-level way to describe the quantum factoring algorithm is that the computation is arranged so that computational paths giving the wrong answer interfere to cancel each other out, leaving a high probability of obtaining the right answer. Another property of quantum systems that plays a crucial role is entanglement, or non-classical correlation, between quantum systems. Many non-quantum physical systems such as waves exhibit interference, but none of these systems exhibits entanglement, and they do not appear usable for quantum computation. Finally, a third property required is the high dimensionality of quantum systems; the dimension of the joint quantum state space of n objects grows exponentially with n, whereas classically the dimension of the joint state space of n objects only grows linearly. The factoring algorithm makes critical use of this extra dimensionality.

In the rest of the paper, I describe these results in more detail. In section 2, I start by discussing Church's thesis, which still appears to hold, and an extension of it, to which quantum computers now appear to be a counterexample. In the following section, I describe the quantum circuit model for quantum computation. This is not laid out particularly well anywhere else, so I spend a reasonable amount of space on it. In section 4, I describe the differences between the quantum circuit model and possible physical realizations of quantum computers, and say a little about why the model appears to give the right definition of what is efficiently computable using quantum mechanics. Section 5 describes the factoring algorithm. Section 6 discusses error-correcting codes and fault-tolerant quantum computing. In the final section, I mention some related results.

2 THE POLYNOMIAL CHURCH'S THESIS.

Church's thesis says that any computable function can be computed on a Turing machine, which is essentially a mathematical abstraction of a digital computer. This thesis arose in the 1930's, and was motivated by the realization that three apparently quite distinct definitions of computable functions were all equivalent. It is well known that Church's thesis is not a theorem, because it does not specify a rigorous mathematical definition of "computable"; specifying such a definition would lead to a provable theorem (and in many cases has), but would also detract from the generality of the thesis. What is somewhat less commonly realized is that this thesis can be viewed as a statement about the laws of physics, simply by interpreting computable to mean computable in the physical world. For this interpretation, if the laws of physics are computable by a Turing machine, then Church's thesis is true.

The development of digital computers rendered the distinction between computable and uncomputable functions too coarse in practice, as it does not take into account the time required for computation. What was needed for the theory of computation was some characterization of efficiently computable functions. In the early 1970's, theoretical computer scientists reached a good compromise between theory and practice with the definition of polynomial-time computable functions. These are functions whose value can be computed in a number of steps polynomial in the input size. The corresponding set of languages—functions whose range is $\{0,1\}$—is known as P (or PTIME). While nobody claims that a function computable in time n^{100} is efficiently computable in practice, the set of polynomial time computable functions is structurally nice enough to use in proofs, and for functions arising in practice it appears to include most of the efficiently computable ones and exclude most of those not efficiently computable. This definition naturally gave rise to a "folk thesis," the polynomial Church's thesis, which says that any function physically computable in time t on some machine X can be computed on a Turing machine in time $p(t)$, where p is a polynomial depending only on the machine X.

Is this folk thesis valid? One good place to start looking for counterexamples is with physical systems which seem to require large amounts of computer time to simulate. Two obvious such candidates are turbulence and quantum mechanics. I will have nothing further to say about turbulence, except that I think the computational complexity of turbulence is a question worthy of serious study. Richard Feynman, in 1982, was the first to consider the case of quantum mechanics [16]. He gave arguments for why quantum mechanical systems should inherently require an exponential overhead to simulate on digital computers. In a lengthy "side remark," he proposed using quantum computers, operating on quantum mechanical principles, to circumvent this problem. David Deutsch [15] followed up on Feynman's proposal by defining quantum Turing machines, and suggesting that if quantum computers could solve quantum mechanical problems more quickly than digital computers, they might also solve classical problems more quickly. It currently appears that this is indeed the case. One piece of evidence for this is that quantum computers can solve certain "oracle problems" faster than classical computers [7, 30]; here an oracle problem is one where the computer is given a subroutine (oracle) which must be treated as a black box. The behavior of computational complexity with respect to oracles, however, has not proved a reliable guide to its true behavior. Another piece of evidence that quantum computers are a counterexample to the polynomial Church's thesis is that they can factor integers and find discrete logarithms in polynomial time, something which it is not known how to do on classical computers despite many years of study. The factorization algorithm is discussed later in this paper.

3 The Quantum Circuit Model.

In this section we discuss the *quantum circuit model* [32] for quantum computation. This is a rigorous mathematical model for a quantum computer. It is not the only mathematical model for quantum computation; there are also the *quantum Turing* *machine model* [7, 32] and the *quantum cellular automata model.* All these models result in the same class of polynomial-time quantum computable functions. Of these, the quantum circuit model is possibly the simplest to describe. It is also easier to connect with possible physical implementations of quantum computers than the quantum Turing machine model. The disadvantage of this model is that it is not naturally a *uniform* model. Uniformity is a technical condition arising in complexity theory, and to make the quantum circuit model uniform, additional constraints must be imposed on it. This issue is discussed later in this section.

In analogy with a classical bit, a two-state quantum system is called a *qubit,* or quantum bit. Mathematically, a qubit takes a value in the vector space $\mathbb{C}^2$. We single out two orthogonal basis vectors in this space, and label these V_0 and V_1. In "ket" notation, which is commonly used in this field, these are represented as $|0\rangle$ and $|1\rangle$. More precisely, quantum states are invariant under multiplication by scalars, so a qubit lives in two-dimensional complex projective space; for simplicity, we work in complex Euclidean space $\mathbb{C}^2$. To conform with physics usage, we treat qubits as column vectors and operate on them by left multiplication.

One of the fundamental principles of quantum mechanics is that the joint quantum state space of two systems is the tensor product of their individual quantum state spaces. Thus, the quantum state space of n qubits is the space $\mathbb{C}^{2^n}$. The basis vectors of this space are parameterized by binary strings of length n. We make extensive use of the tensor decomposition of this space into n copies of $\mathbb{C}^2$, where $V_{b_1 b_2 \cdots b_n} = V_{b_1} \otimes V_{b_2} \otimes \ldots \otimes V_{b_n}$. Generally, we use position to distinguish the n different qubits. Occasionally we need some other notation for distinguishing them, in which case we denote the i'th qubit by $V^{[i]}$. Since quantum states are invariant under multiplication by scalars, they can be normalized to be unit length vectors; except where otherwise noted, quantum states in this paper are normalized. Quantum computation takes place in the quantum state space of n qubits $\mathbb{C}^{2^n}$, and obtains extra computational power from its exponential dimensionality.

In a usable computer, we need some means of giving it the problem we want solved (input), some means of extracting the answer from it (output), and some means of manipulating the state of the computer to transform the input into the desired output (computation). We next briefly describe input and output for the quantum circuit model. We then take a brief detour to describe the classical circuit model; this will motivate the rules for performing the computation on a quantum computer.

Since we are comparing quantum computers to classical computers, the input to a quantum computer will be classical information. It can thus can be expressed as a binary string S of some length k. We need to encode this in the initial quantum state of the computer, which must be a vector in $\mathbb{C}^{2^n}$. The way we do this is to concatenate the bit string S with $n-k$ 0's to obtain the length n string $S0\ldots0$. We then initialize the quantum computer in the state $V_{S0\ldots0}$. Note that the number of qubits is in general larger than the input. These extra qubits are often required as workspace in implementing quantum algorithms.

At the end of a computation, the quantum computer is in a state which is a unit vector in $\mathbb{C}^{2^n}$. This state can be written explicitly as $W = \sum_s \alpha_s V_s$ where s ranges over binary strings of length n, $\alpha_s \in \mathbb{C}$, and $\sum_s |\alpha_s|^2 = 1$. These α_s

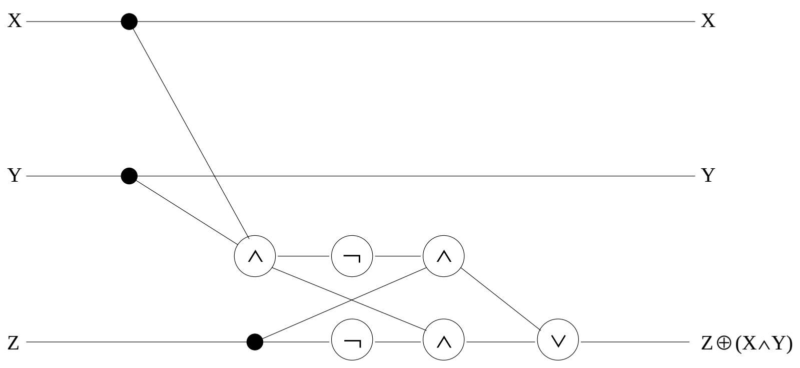

Figure 1: Construction of a Toffoli gate using the classical gates AND, OR and NOT. The input is on the left and the output on the right.

are called *probability amplitudes,* and we say that W is a *superposition* of basis vectors V_s. In quantum mechanics, the Heisenberg uncertainty principle tells us that we cannot measure the complete quantum state of this system. There are a large number of permissible measurements; for example, any orthogonal basis of $\mathbb{C}^{2^n}$ defines a measurement whose possible outcomes are the elements of this basis. However, we assume that the output is obtained by projecting each qubit onto the basis $\{V_0, V_1\}$. When applied to a state $\sum_s \alpha_s V_s$, this projection produces the string s with probability $|\alpha_s|^2$. The quantum measurement process is inherently probabilistic. Thus we do not require that the computation gives the right answer all the time; but that we obtain the right answer at least 2/3 of the time. Here, the probability 2/3 can be replaced by any number strictly between 1/2 and 1 without altering what can be computed in polynomial time by quantum computers—if the probability of obtaining the right answer is strictly larger than 1/2, it can be amplified by running the computation several times and taking the majority vote of the results of these separate computations.

In order to motivate the rules for state manipulation in a quantum circuit, we now take a brief detour and describe the classical circuit model. Recall that a classical circuit can always be written solely with the three gates AND ($\wedge$), OR ($\vee$) and NOT ($\neg$). These three gates are thus said to form a *universal* set of gates. Figure 1 gives an example circuit for a computation called a *Toffoli gate* using these three types of gates. Besides these three gates, note that we also need elements which duplicate the values on wires. These duplicating "gates" are not possible in the domain of quantum computing.

A quantum circuit is similarly built out of logical quantum wires carrying qubits, and quantum gates acting on these qubits. Each wire corresponds to one of the n qubits. We assume each gate acts on either one or two wires. The possible physical transformations of a quantum system are unitary transformations, so each quantum gate can be described by a unitary matrix. A quantum gate on one qubit is then described by a 2×2 matrix, and a quantum gate on two qubits by a 4×4 matrix. Note that since unitary matrices are invertible, the computation is reversible; thus starting with the output and working backwards one obtains the input. Further note that for quantum gates, the dimension of the output space is equal to that of the input space, so at all times during the computation we have n qubits carried on n quantum wires. Figure 2 contains an example of a quantum circuit for computing a Toffoli gate.

Quantum gates acting on one or two qubits ($\mathbb{C}^2$ or $\mathbb{C}^4$) naturally induce a transformation on the state space of the entire quantum computer ($\mathbb{C}^{2^n}$). For example, if A is a 4×4 matrix acting on qubits i and j, the induced action on a basis vector of $\mathbb{C}^{2^n}$ is

$$A^{[i,j]}\, V_{b_1 b_2 \cdots b_n} = \sum_{s=0}^{1} \sum_{t=0}^{1} A_{b_i b_j\ st}\, V_{b_1 b_2 \cdots b_{i-1} s b_{i+1} \cdots b_{j-1} t b_{j+1} \cdots b_n}. \tag{1}$$

This is a tensor product of A (acting on qubits i and j) with the identity matrix (acting on the remaining qubits). When we multiply a general vector by a quantum gate, it can have negative and positive coefficients which cancel out, leading to quantum interference.

As there are for classical circuits, there are also universal sets of gates for quantum circuits; such a universal set of gates is sufficient to build circuits for any quantum computation. One particularly useful universal set of gates is the set of all one-bit gates and a specific two-bit gate called the Controlled NOT (CNOT). These gates can efficiently simulate any quantum circuits whose gates act on only a constant number of qubits [2]. On basis vectors, the CNOT gate negates the second (target) qubit if and only if the first (control) qubit is 1. In other words, it takes V_{XY} to V_{XZ} where $Z = X + Y \pmod 2$. This corresponds to the unitary matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2}$$

Note that the CNOT is a classical reversible gate. To obtain a universal set of classical reversible gates, you need at least one reversible three-bit gate, such as a Toffoli gate; otherwise you can only perform linear Boolean computations. A Toffoli gate is a doubly controlled NOT, which negates the 3rd bit if and only if the first two are both 1. By itself the Toffoli gate is universal for reversible classical computation, as it can simulate both AND and NOT gates [17]. Thus, if you can make a Toffoli gate, you can perform any reversible classical computation. Further, as long as the input is not erased, any classical computation can be efficiently performed reversibly [3], and thus implemented efficiently by Toffoli gates.

Because of the extra possibilities allowed by quantum interference, for quantum circuits the CNOT together with all quantum one-bit gates forms a universal set of gates. Figure 2 gives a construction of a Toffoli gate out of CNOT gates and one-bit gates [2], showing that this set is at least universal for classical computation. This particular construction does not result in a Toffoli gate with all positive

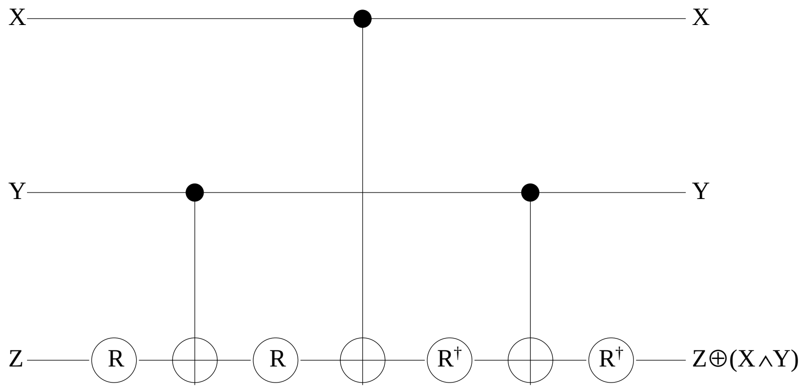

Figure 2: Construction of a Toffoli gate using quantum gates. The gates represented by $\oplus$ are CNOT's, where the circle identifies the target qubit. The gate R is $\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$, and $R^\dagger$ is the Hermitian transpose of R. In this construction, the phase on V_{101} is -1, and all the other phases are $+1$; the phases can all be made $+1$ by a somewhat more complicated quantum circuit.

phases—multiplying the corresponding matrices in Figure 2 produces the matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{3}$$

which is the classical Toffoli gate with a phase of -1 on one of its outcomes. This still acts classically as a Toffoli gate, since phases are irrelevant to classical computation. In quantum computation, however, we must keep careful track of phases. A more complicated circuit can be constructed which eliminates this phase of -1 [2].

We now define the complexity class BQP, which stands for bounded-error quantum polynomial time. This is the class of languages which can be computed on a quantum computer in polynomial time, with the computer giving the correct answer at least 2/3 of the time. To give a rigorous definition of this complexity class using quantum circuits, we need to consider uniformity conditions. Any specific quantum circuit can only compute a function whose domain (input) is binary strings of a specific length. To use the quantum circuit model to implement functions taking arbitrary length binary strings as input, we take a family of quantum circuits, containing one circuit for inputs of each length. Without any further conditions on the family of circuits, the designer of this circuit family could hide an uncomputable function in the design of the circuits for each input length.

This definition would thus result in the unfortunate inclusion of uncomputable functions in the complexity class BQP. To exclude this possibility, we require *uniformity* conditions on the circuit family. The easiest way of doing this is to require a classical Turing machine that on input n outputs a description of the circuit for length n inputs, and which runs in time polynomial in n. For quantum computing, we need an additional uniformity condition on the circuits. It would be possible for the circuit designer to hide uncomputable (or hard-to-compute) information in the unitary matrices corresponding to quantum gates. We thus require that the k'th digit of the entries of these matrices can be computed by a second Turing machine in time polynomial in k. Although we do not have space to discuss this fully, the power of the machines designing the circuit family can actually be varied over a wide range; this helps us convince ourselves that we have the right definition of BQP.

The definition of polynomial time computable functions on a quantum computer is thus those functions computable by a *uniform* family of circuits whose size (number of gates) is polynomial in the length of the input, and which for any input gives the right answer at least 2/3 of the time. The corresponding set of languages (functions with values in $\{0, 1\}$) is called BQP.

4 RELATION OF THE MODEL TO QUANTUM PHYSICS.

The quantum circuit model of the previous section is much simplified from the realities of quantum physics. There are operations possible in physical quantum systems which do not correspond to any simple operation allowable in the quantum circuit model, and complexities that occur when performing experiments that are not reflected in the quantum circuit model. This section contains a brief discussion of these issues, some of which are discussed more thoroughly in [7].

In everyday life, objects behave very classically, and on large scales we do not see any quantum mechanical behavior. This is due to a phenomenon called decoherence, which makes superpositions of states decay, and makes large-scale superpositions of states decay very quickly. A thorough discussion of decoherence can be found in [35]; one reason it occurs is that we are dealing with open systems rather than closed ones. Although closed systems quantum mechanically undergo unitary evolution, open systems need not. They are subsystems of systems undergoing unitary evolution, and the process of taking subsystems does not preserve unitarity.

However hard we may try to isolate quantum computers from the environment, they will still undergo some decoherence and errors. We need to know that these processes do not fundamentally change their behavior. Using no error correction, if each gate results in an amount of decoherence and error of order $1/t$, then $O(t)$ operations can be performed before the quantum state becomes so noisy as to usually give the wrong answer [7]. Active error correction can improve this situation substantially, and is discussed in section 6.

In some proposed physical architectures for quantum computers, there are restrictions that are more severe than the quantum computing model. Many of these restrictions do not change the class BQP. For example, it could easily be the case that a gate could only be applied to a pair of *adjacent* qubits. We can still operate on a pair of arbitrary qubits: by repeatedly exchanging one of these qubits with a neighbor we can bring this pair together. If there are n qubits in the computer, this can only increase the computation time by a factor of n, preserving the complexity class BQP.

The quantum circuit model described in the previous section postpones all measurements to the end, and assumes that we are not allowed to use probabilistic steps. Both of these possibilities are allowed in general by quantum mechanics, but neither possibility makes the complexity class BQP larger [7]. For fault-tolerant quantum computing, however, it is very useful to permit measurements in the middle of the computation, in order to measure and correct errors.

The quantum circuit model also assumes that we only operate on a constant number of qubits at a time. In general quantum systems, all the qubits evolve simultaneously according to some Hamiltonian describing the system. This simultaneous evolution of many qubits cannot be described by a single gate in our model, which only operates on two qubits at once. In a realistic model of quantum computation, we cannot allow general Hamiltonians, since they are not experimentally realizable. Some Hamiltonians that act on all the qubits at once, however, are experimentally realizable. It would be nice to know that even though these Hamiltonians cannot be directly described by our model, they cannot be used to compute functions not in BQP in polynomial time. This could be accomplished by showing that systems with such Hamiltonians can be efficiently simulated by a quantum computer. Some work has been done on simulating Hamiltonians on quantum computers [1, 24, 33], but I do not believe this question has been completely addressed yet.

An important aspect of quantum mechanics not used in the quantum circuit model is that identical particles are indistinguishable; in general they must obey either Fermi-Dirac or Einstein-Bose statistics when they are interchanged. Particle statistics do not appear to add any power to the quantum computing model, but I do not believe this has been rigorously proved.

From the view of the current state of experimental physics, quantum computers appear to be extremely difficult to build, but do not seem to violate any fundamental physical laws. As qubits, we need to use quantum systems which are relatively stable, which interact strongly with each other (to carry out quantum gates quickly), but which interact weakly with everything else (to avoid errors caused by interaction with the environment). Since the discovery of the factoring algorithm, a variety of proposals for experimental implementation of quantum computers have been made. One of these proposals is to use the electronic states of ions in an electromagnetic ion trap as the qubits, to manipulate them using lasers, and to communicate between different ions using a vibrational mode of the ions, or *phonon* [12]. Another is to use nuclear spins of atoms in a complex molecule as the qubits, and to manipulate them using nuclear magnetic resonance spectroscopy [14, 18]. A quite recent proposal is to use nuclear spins of impurities embedded in a silicon chip as the qubits, and to manipulate them using electronics on the same chip [23]. None of these proposals has been experimentally realized for more than a handful of qubits, but they all have proponents who believe that they may be scaled up to obtain much larger working quantum computers.

5 The Factoring Algorithm.

For factoring an L-bit number N, the best classical algorithm known is the number field sieve, which asymptotically takes time $O(\exp(cL^{1/3}\log^{2/3}L))$. On a quantum computer, the quantum factoring algorithm takes asymptotically $O(L^2 \log L \log\log L)$ steps. The key idea of the quantum factoring algorithm is the use of a Fourier transform to find the period of the sequence $u_i = x^i \pmod N$, from which period a factorization of N can be obtained. The period of this sequence is exponential in L, so this approach is not practical on a digital computer. On a quantum computer, however, we can find the period in polynomial time by exploiting the 2^{2L}-dimensional state space of $2L$ qubits, and taking a Fourier transform over this space. The exponential dimensionality of this space permits us to take the Fourier transform of an exponential length sequence. How this works should be clearer from the following sketch of the algorithm, the full details of which are in [28], along with a quantum algorithm for finding discrete logarithms.

The idea behind all the fast factoring algorithms (classical or quantum) is fairly simple. To factor N, find two residues mod N such that

$$s^2 \equiv t^2 \pmod N \tag{4}$$

but $s \not\equiv \pm t \pmod N$. We now have

$$(s+t)(s-t) \equiv 0 \pmod N \tag{5}$$

and neither of these two factors is 0 (mod N). Thus, $s+t$ must contain one factor of N (and $s-t$ another). We can extract this factor by finding the greatest common divisor of $s+t$ and N; this computation can be done in polynomial time using Euclid's algorithm.

In the quantum factoring algorithm, we find the multiplicative period r of a residue $x \pmod N$. This period r satisfies $x^r \equiv 1 \pmod N$; if we are lucky then r is even, so both sides of this congruence are squares, and we can try the above factorization method. If we have just a little bit more luck, then $x^{r/2} \not\equiv -1 \pmod N$, so we obtain a factor by computing $\gcd(x^{r/2}+1, N)$. It is a fairly simple exercise in number theory to show that for large N with two or more prime factors, at least half the residues $x \pmod N$ produce prime factors using this technique, and that for most large N, the fraction of good residues x is much higher; thus, if we try several different values for x, we have to be particularly unlucky not to obtain a factorization using this method.

We now need to explain what the quantum Fourier transform is. The quantum Fourier transform on k qubits maps the state V_a, where a is considered as an integer between 0 and $2^k - 1$, to a superposition of the states V_b as follows:

$$V_a \to \frac{1}{2^{k/2}} \sum_{b=0}^{2^k-1} \exp(2\pi i ab/2^k)\, V_b \tag{6}$$

It is easy to check that this transformation defines a unitary matrix. It is not as straightforward to implement this Fourier transform as a sequence of one- and two-bit quantum gates. However, an adaption of the Cooley-Tukey algorithm decomposes this transformation into a sequence of $k(k-1)/2$ one- and two-bit gates. More generally, the discrete Fourier transform over any product Q of small primes (of size at most $\log Q$) can be performed in polynomial time on a quantum computer.

We are now ready to give the quantum algorithm for factoring. What we do is design a polynomial-size circuit which starts in the quantum state $V_{00\ldots0}$ and whose output, with reasonable probability, lets us factor an L-bit number N in polynomial time using a digital computer. This circuit has two main registers, the first of which is composed of $2L$ qubits and the second of L qubits. It also requires a few extra qubits of work space, which we do not mention in the summary below but which are required for implementing the step (8) below.

We start by putting the computer into the state representing the superposition of all possible values of the first register:

$$\frac{1}{2^L}\sum_{a=0}^{2^{2L}-1} V_a \otimes V_0. \tag{7}$$

This can easily be done using $2L$ gates by putting each of the qubits in the first register into the state $\frac{1}{\sqrt{2}}(V_0 + V_1)$.

We next use the value of a in the first register to compute the value $x^a \pmod N$ in the second register. This can be done using a reversible classical circuit for computing $x^a \pmod N$ from a. Computing $x^a \pmod N$ using repeated squaring takes asymptotically $O(L^3)$ quantum gates using the grade school multiplication algorithm, and $O(L^2 \log L \log\log L)$ gates using fast integer multiplication (which is actually faster only for relatively large values of L). This leaves the computer in the state

$$\frac{1}{2^L}\sum_{a=0}^{2^{2L}-1} V_a \otimes V_{x^a \pmod N}. \tag{8}$$

The next step is to take the discrete Fourier transform of the first register, as in Equation (6). This puts the computer into the state

$$\frac{1}{2^{2L}}\sum_{a=0}^{2^{2L}-1}\sum_{b=0}^{2^{2L}-1} \exp(2\pi i ab/2^{2L}) V_b \otimes V_{x^a \pmod N}. \tag{9}$$

Finally, we measure the state. This will give the output $V_b \otimes V_{x^j \pmod N}$ with probability equal to the square of the coefficient on this vector in the sum (9). Since many values of $x^a \pmod N$ are equal, many terms in this sum contribute to each coefficient. Explicitly, this probability is:

$$\frac{1}{2^{4L}}\left|\sum_{\substack{a\equiv j \pmod r\\ 0\le a<2^{2L}}}\ \sum_{b=0}^{2^{2L}-1} \exp(2\pi i ab/2^{2L})\right|^2. \tag{10}$$

This is a geometric sum, and it is straightforward to check that this sum is very small except when

$$rb \approx d2^{2L} \tag{11}$$

for some integer d. We thus are likely to observe only values of b satisfying (11). Rewriting this equation, we obtain

$$\frac{b}{2^{2L}} \approx \frac{d}{r}. \tag{12}$$

We know b and 2^{2L}, and we want to find r. We chose $2L$ as the size of the first register in order to make d/r likely to be the closest fraction to $b/2^{2L}$ with denominator at most N. Thus, all we need do to find r is to round $b/2^{2L}$ to a fraction with denominator less than N. This can be done in polynomial time using a continued fraction expansion.

More details of this algorithm can be found in [28]. Recently, Zalka [34] has analyzed the resources required by this algorithm much more thoroughly, improving upon their original values in many respects. For example, he shows that you can use only $3L + o(L)$ qubits, whereas the original algorithm required $2L$ extra qubits for workspace, giving a total of $5L$ qubits. He also shows how to efficiently parallelize the algorithm to run on a parallel quantum computer.

6 Quantum Error Correcting Codes.

One of the reactions to the quantum factoring paper was that quantum computers would be impossible to build because it would be impossible to reduce decoherence and errors to levels low enough to ensure reliable quantum computation. Indeed, without error correction, it would probably be an impossible task to build quantum computers large enough to factor 100-digit numbers—factoring such a number requires billions of steps, so each step would need to be accurate to better than one part in a billion, a virtually impossible challenge in experimental physics. Fortunately, it is possible to design fault-tolerant circuits for quantum computers, which allow computations of arbitrary length to be performed with gates having accuracy of only some constant c. Current estimates using known methods for constructing fault-tolerant quantum circuits put this constant in the range of 10^{-4} [25]; improved techniques could increase this value.

For some time after the factoring algorithm was discovered, however, it was believed that making quantum computers fault-tolerant was impossible. There were a number of plausible arguments for why this should be true. One argument for the impossibility of quantum error correction was based on the theorem, related to the Heisenberg uncertainty principle, that an unknown quantum state cannot be duplicated. The argument was that since you cannot duplicate quantum information, you cannot have more than one copy of a qubit around at any given time, and thus that it was impossible to protect a qubit from errors. Indeed, the simplest classical error correcting code is the 3-repetition code, which triplicates each bit, and other classical error correcting codes also appear to be based on repetition. Despite this pessimistic argument, quantum error correcting codes do exist, and are generalizations of classical error-correcting codes. The codes protect quantum information from error and decoherence not by duplicating it, but by hiding it in subspaces of $\mathbb{C}^{2^n}$ which are affected very little by decoherence and errors that act on only one qubit, or only a small number of qubits.

Before we discuss quantum error correcting codes in detail, we need to say more about the measurement process. For every set of orthogonal subspaces of C^{2^n} which span the entire space, there is a measurement which outputs one of these subspaces as classical data, and which projects the original quantum state onto this subspace. For example, if our quantum state is $\sum_{s=0}^{2^n-1} \alpha_s V_s$ and we measure the first qubit, we obtain the (not normalized) quantum state

$$\sum_{s'=0}^{2^{n-1}-1} \alpha_{0s'} V_{0s'} \quad \text{with probability} \quad \sum_{s'=0}^{2^{n-1}-1} |\alpha_{0s'}|^2, \tag{13}$$

and the state

$$\sum_{s'=0}^{2^{n-1}-1} \alpha_{1s'} V_{1s'} \quad \text{with probability} \quad \sum_{s'=0}^{2^{n-1}-1} |\alpha_{1s'}|^2. \tag{14}$$

This measurement corresponds to the partition of $\mathbb{C}^{2^n}$ into the two subspaces generated by $\{V_{0s'}\}$ and by $\{V_{1s'}\}$.

To illustrate how quantum error correcting codes work, we first explain what goes wrong when we try to extend the straightforward repetition code to the quantum realm. The obvious thing to do is to take

$$\begin{array}{ccc} V_0 & \rightarrow & V_{000} \\ V_1 & \rightarrow & V_{111} \end{array} \tag{15}$$

This indeed does protect against value errors in our qubits. Suppose we apply the error transformation $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ to the first qubit. Then the encodings of V_0 and V_1 get taken to the states V_{100} and V_{011}, respectively. The subspace generated by these two quantum states is orthogonal to that generated by the original codewords V_{000} and V_{111}. We can thus make a measurement which reveals that there was an bit flip in the first qubit without measuring (and thus disturbing) the encoded quantum state. It is easily seen that bit flips applied to each of the three qubits create subspaces that are orthogonal to each other, so there is a quantum measurement which identifies on which qubit a bit flip error occurred without disturbing the encoded state. It is then straightforward to fix the bit flip error by applying a quantum gate to the qubit in error.

However, a phase error on one the qubits is disastrous in this code. What happens when the error transformation $\begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}$ is applied to one of the qubit is that it takes an encoded V_0 to an encoded V_0, and takes an encoded V_1 to an encoded $e^{i\phi} V_1$. Thus, a phase error an *any* of the three qubits translates to a phase error on the encoded qubit, making the encoding three times as vulnerable to phase errors.

We now explain the above difficulty another way which illuminates the construction of quantum error-correcting codes. We consider phase flip errors, which are phase errors with $\phi = \pi$. There is a transformation that takes phase flips to bit flips and vice versa. This is the Hadamard transform, which is

$$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{16}$$

When this is applied to all the qubits in the code above, as well as the encoded qubits, we get the code

$$\begin{aligned} V_0 &\rightarrow \frac{1}{2}(V_{000} + V_{110} + V_{101} + V_{011}) \\ V_1 &\rightarrow \frac{1}{2}(V_{111} + V_{001} + V_{010} + V_{100}) \end{aligned} \tag{17}$$

Notice that for this code, a single bit flip interchanges V_0 and V_1, so this code cannot correct bit flips, again showing that code (15) cannot correct phase flips.

What we need to make a good quantum error correcting code is a code having the property that bit flips can be corrected both before and after the application of the Hadamard transformation. Such a code can be found by generalizing the codes (15) and (17), and it was discovered independently by two groups [11, 31]. It is based on the classical 7-bit Hamming code, and is defined as follows:

$$\begin{aligned} V_0 \rightarrow\ & \frac{1}{\sqrt{8}}\big(V_{0000000} + V_{1110100} + V_{0111010} + V_{0011101} \\ & + V_{1001110} + V_{0100111} + V_{1010011} + V_{1101001}\big) \\ V_1 \rightarrow\ & \frac{1}{\sqrt{8}}\big(V_{1111111} + V_{0001011} + V_{1000101} + V_{1100010} \\ & + V_{0110001} + V_{1011000} + V_{0101100} + V_{0010110}\big). \end{aligned} \tag{18}$$

The indices of the basis vectors in the support of the encoded states are exactly the classical 7-bit Hamming code. The fact that the classical Hamming code corrects one error means this code can correct one bit flip. This quantum code is taken to itself under the application of the Hadamard transform (16) both to the encoded qubit and to each encoding qubit, showing that it is also able to correct one phase flip. In fact, it can correct a simultaneous bit flip and phase flip.

We now have a seven bit code that can corrects a phase and/or a bit flip applied to one of its qubits. This is by no means the complete set of possible quantum mechanical errors on one qubit; this set is parameterized by several continuous variables. However, the ability of a quantum code to correct the following set of four one-bit errors confers on it the ability to correct any possible one-bit quantum error:

$$\mathbf{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},\ \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},\ \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},\ \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}. \tag{19}$$

These four errors correspond to no error, a bit flip, a phase flip, and a simultaneous bit and phase flip, respectively. We do not have enough space to explain this in detail, but the fact that these form a basis for the set of 2×2 matrices is enough to imply they can correct any one-bit quantum error. The rigorous details of this implication are in [11]; a more intuitive explanation is in [6].

This quantum Hamming code is the smallest nontrivial example of a set of codes based on linear binary codes named CSS codes after their discoverers [11, 31], and which contains codes that are much more efficient than this first one. For fault tolerance, which will be discussed next, we only need to use CSS codes. However, a more general framework that includes these codes was discovered simultaneously by two groups [19, 20, 10]. Substantial work on quantum error correcting codes has occurred since their discovery, much of it referenced in [10].

In classical computers, error correcting codes have been found to be very useful for storing and transmitting information, but not for providing fault-tolerant computing. It is difficult to perform gates on encoded qubits, and once the qubits have been decoded, they are no longer protected from error. Theoretically, the best way to provide high levels of fault tolerance for classical circuits was discovered by von Neumann, who discovered it after reasoning that some means of protection from error had to exist in biological systems. This method involves the use of massive redundancy. If you plan to run your computer for t steps, you make $c \log t$ copies of every bit, and during the computation, you continually compare them in order to catch any errors you have made. The drawback of this method is that it requires $c \log t$ overhead, which is too expensive for use in practice, given the remarkably low levels of error obtainable by current computer hardware. On the other hand, it can be shown that if you must use unreliable gates, $O(\log t)$ overhead is required to achieve reliable computation, so von Neumann's construction is up to a constant factor best possible.

As in classical computers, quantum error correcting codes should work well for protecting qubits while they are being stored and transmitted. However, because quantum data cannot be cloned, fault tolerance using massive redundancy cannot work in quantum computers. We thus need another method. The methods currently known for providing fault tolerance in quantum computers are based on quantum error correcting codes [25, 29]. To use quantum error correcting codes for reliable computation, we need to show how to do two additional things with them, neither of which is at first glance obviously possible. These are:

1. correct errors using noisy gates so that errors are corrected faster than new errors are introduced;
2. perform quantum gates on encoded bits without decoding them, while making sure that any errors cannot propagate too widely during the computation.

We do not have much space to discuss how to accomplish these tasks, so we say nothing about the first task, and give only a very broad sketch of how the second task can be accomplished.

In order to compute on encoded qubits without decoding, we need fault-tolerant implementations of a universal set of quantum gates on the encoded qubits. What we need are circuits having the property that if errors occur in only a few quantum gates, or are present in a few of the inputs, these errors cannot affect too many of the qubits in the output of the gate (otherwise, there will

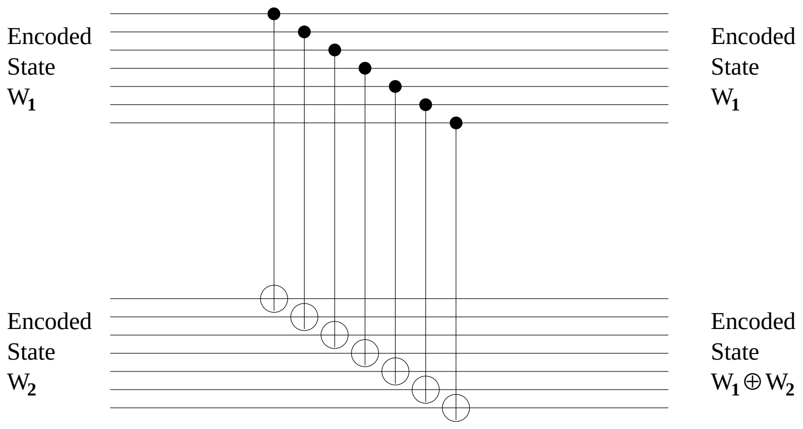

Figure 3: Implementation of a CNOT gate on qubits encoded using the quantum Hamming code (18). This circuit can be used in fault-tolerant quantum circuits, since an error in the i'th wire of an encoded qubit (or in the i'th gate) can only propagate to the i'th wire of each of the output qubits. Gates that are implementable on encoded qubits in this fashion are called *transversal* gates.

be more errors than the quantum error-correcting codes we are using can correct). It turns out that certain gates are easy to implement this way. Figure 3 shows how to perform a CNOT on two encoded qubits by performing it on each pair of encoding wires. Similarly, if a Hadamard gate (16) is applied to each quantum wire, a Hadamard gate is performed on the encoded qubit. Implementations of this type are called transversal gates, and these do not form a universal set of quantum gates. We need to supplement the set of transversal gates with an extra gate implemented using another method. It was shown how to perform the Toffoli gate fault-tolerantly on encoded qubits in [29], and the set of transversal gates augmented by this gate is a universal set of gates.

To implement a circuit of size t fault-tolerantly, the techniques of [29] required gates with error rate at most $O(1/(\log t)^c)$. To obtain fault tolerance using gates with constant error rate requires a further idea: the use of concatenated codes. These are nested codes, where each layer catches most of the errors missed by the previous layer. Judicious use of concatenated codes and careful analysis shows that gates with some constant error rate are able to produce fault-tolerant quantum circuits; this constant is currently estimated at around 10^{-4}. For more details, see the excellent survey of fault-tolerance in quantum computing [25].

7 Other Work.

This section discusses areas related to quantum computing; it is not intended to be a complete survey, but a somewhat idiosyncratic view of some results I find interesting. I have tried to mention survey articles when they exist, so the interested reader can find pointers to the literature. One excellent resource is the quant-ph preprint archive, at `http://xxx.lanl.gov/`, containing preprints of many recent research articles in this field. John Preskill, at Caltech, recently taught a course on quantum computing and quantum information, and his excellent set of lecture notes is available on the web [26].

As Feynman suggested, it appears that quantum computing is good at computing simulations of quantum mechanical dynamics. Some work has already appeared showing this [1, 24, 33], but much remains to be done.

A significant algorithm in quantum computing is L. K. Grover's search algorithm, which searches an unordered list of N items (or the range of an efficiently computable function) for a specific item in time $O(\sqrt{N})$, an improvement on the optimal classical algorithm, which must look at $N/2$ items on average before finding a specific item [21]. The technique used in this algorithm can be applied to a number of other problems to also obtain a square root speed-up [22].

One of the earliest applications of quantum mechanics to areas related to computing is quantum cryptography, more specifically quantum key distribution. Consider two people trying to share some secret information which they can then use as a key for a cryptosystem. If they can only communicate over a phone line possibly open to eavesdroppers, they have no choice but to use public key cryptography [27], which may be open to attack by a quantum computer or (say) discovery of a fast factoring algorithm on a classical computer. However, if they in addition have access to an optical fiber which they can use to transmit quantum states, they can use quantum cryptography [4]. One of them (the sender) transmits states chosen at random from a set of non-orthogonal quantum states (e.g. V_0, V_1, $\frac{1}{\sqrt{2}}(V_0 + V_1)$, $\frac{1}{\sqrt{2}}(V_0 - V_1)$) The receiver then reads the states in either the basis $\{V_0, V_1\}$ or $\{\frac{1}{\sqrt{2}}(V_0 \pm V_1)\}$, again chosen at random. Communicating over a classical channel using a special protocol, they can figure out the states for which they agree on the measurement basis; they should agree on about half the states, each of which supplies a bit towards a secret key. If an eavesdropper was listening, she cannot have gained too much information—since she does not know in which basis the states were transmitted, any information she gains must cause a disturbance in the states, which the sender and receiver can detect by measuring some of their states instead of using them for the secret key. They can also further sacrifice some of their bits to ensure that the eavesdropper gains virtually no information about the remaining bits of their key, and that they agree on all the bits of this key. Since the original quantum cryptography papers, there have been many articles either proposing other schemes or working towards rigorous proofs that the scheme is secure against all possible quantum attacks (i.e., when the eavesdropper has access to a quantum computer). A good bibliography on quantum cryptography is [8].

Quantum cryptography is but one aspect of a rapidly burgeoning subject, quantum information theory. A startling result in this field, the interest in which helped contribute to its recent rapid growth, was the discovery of quantum teleportation [5]. It is not possible to transmit an unknown quantum state using only classical information (say, over a telephone line). However, if two people share an EPR pair, such as the quantum state $\frac{1}{\sqrt{2}}(V_{01} - V_{10})$, with the sender holding the first qubit and the receiver holding the second, then they can transmit an unknown quantum bit using a classical channel. The sender performs a combined measurement on the unknown state and the EPR pair, and transmits the classical two-bit outcome to the receiver, who then uses this information to reconstruct the unknown state from his half of the EPR pair. The act of teleportation thus uses up the resource of *entanglement* between the sender and the receiver, which is present in the EPR pair. One research direction in quantum information theory is quantifying the amount of entanglement in a quantum state. Another direction is measuring the classical and the quantum capacities of a quantum channel. More information on quantum information theory can be found in Preskill's course notes [26] and in the survey article [6].

Another recent development is the study of quantum communication complexity. If two people share quantum entanglement, as well as a classical communications channel, this permits them to send each other qubits, but does not reduce the number of bits required for transmission of classical information. However, if they both have some classical data, and they wish to compute some classical function of this data, shared quantum entanglement may help reduce the amount of purely classical communication required to compute this function. This was first shown by Cleve and Burhman [13]. More results on communication complexity have since been shown, and some of these were recently used to give lower bounds on the power of quantum computers in the black-box (oracle) model [9].

There has been a substantial amount of recent work on both quantum error correcting codes and quantum fault tolerance. Many results on quantum error correcting codes are reviewed in [10], and Preskill has written an excellent survey of fault tolerant quantum computing [25].

References

[1] D. S. Abrams and S. Lloyd, Simulation of many-body Fermi systems on a universal quantum computer, *Phys. Rev. Lett.* 79 (1997), 2586-2589.

[2] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, Elementary gates for quantum computation, *Phys. Rev. A* 52 (1995), 3457–3467.

[3] C. H. Bennett, Logical reversibility of computation, *IBM J. Res. Develop.* 17 (1973), 525–532.

[4] C. H. Bennett, G. Brassard, Quantum cryptography: public key distribution and coin tossing, in *Proc. IEEE International Conference on Computers, Systems and Signal Processing,* Bangalore, India (1984) 175–179.

[5] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels, *Phys. Rev. Lett.* 70 (1993), 1895–1898.

[6] C. H. Bennett and P. W. Shor, Quantum information theory, *IEEE Transactions on Information Theory* (1998), to appear.

[7] E. Bernstein and U. Vazirani, Quantum complexity theory, *SIAM J. Computing* 26 (1997), 1411–1473.

[8] G. Brassard, Quantum cryptography: a bibliography, *SIGACT News* 24:3 (1993), 16–20. A more recent version is online at:
`http://www.iro.umontreal.ca/~crepeau/Biblio-QC.html`.

[9] H. Burhman, R. Cleve and A. Wigderson, Quantum vs. classical communication and computation, in *Proceedings of the 30th Annual ACM Symposium on Theory of Computing,* ACM Press, New York (1998), 63–69.

[10] A. R. Calderbank, E. M. Rains, P. W. Shor and N. J. A. Sloane, Quantum error correction via codes over GF(4), *IEEE Transactions on Information Theory* 44 (1998), 1369–1387.

[11] A. R. Calderbank and P. W. Shor, Good quantum error-correcting codes exist, *Phys. Rev. A* 54 (1995) 1098-1106.

[12] J. I. Cirac and P. Zoller, Quantum computations with cold trapped ions, *Phys. Rev. Lett.* 74 (1995), 4091–4094.

[13] R. Cleve and H. Burhman, Substituting quantum entanglement for communication, *Phys. Rev. A* 56 (1997), 1201–1204.

[14] D. G. Cory, A. F. Fahmy and T. F. Havel, Ensemble quantum computing by nuclear magnetic resonance spectroscopy, *Proc. Nat. Acad. Sci.* 94 (1997), 1634–1639.

[15] D. Deutsch, Quantum theory, the Church–Turing principle and the universal quantum computer, *Proc. Roy. Soc. London Ser. A* 400 (1985) 96–117.

[16] R. Feynman, Simulating physics with computers, *Internat. J. Theoret. Phys.* 21 (1982), 467–488.

[17] E. Fredkin and T. Toffoli, Conservative logic, *Internat. J. Theoret. Phys.* 21 (1982), 219–253.

[18] N. A. Gershenfeld and I. L. Chuang, Bulk spin resonance quantum computation *Science* 275 (1997), 350-356.

[19] D. Gottesman, A class of quantum error-correcting codes saturating the quantum Hamming bound, *Phys. Rev. A* 54 (1996), 1862–1868.

[20] D. Gottesman, *Stabilizer Codes and Quantum Error Correction,* Ph.D. Thesis, California Institute of Technology (1997). Also LANL e-print quant-ph/9705052, available online at `http://xxx.lanl.gov/`.

[21] L. K. Grover, Quantum mechanics helps in searching for a needle in a haystack, *Phys. Rev. Lett.* 78 (1997), 325–328.

[22] L. K. Grover, A framework for fast quantum mechanical algorithms, in *Proceedings of the 30th Annual ACM Symposium on Theory of Computing,* ACM Press, New York (1998), 53–62.

[23] B. E. Kane, A silicon-based nuclear spin quantum computer, *Nature* 393 (1998), 133–137.

[24] S. Lloyd, Universal quantum simulators, *Science* 273 (1996), 1073-1078.

[25] J. Preskill, Fault-tolerant quantum computation, to appear in *Introduction to Quantum Computation,* H.-K. Lo, S Popescu and T. P. Spiller, eds. (1998). Also LANL e-print quant-ph/9712048, available online at `http://xxx.lanl.gov/`.

[26] J. Preskill, Lecture notes for Physics 229 (1998), available online at `http://www.theory.caltech.edu/people/preskill/ph229/`.

[27] R. L. Rivest, A. Shamir and L. Adleman, A method of obtaining digital signatures and public-key cryptosystems, *Comm. Assoc. Comput. Mach.* 21 (1978), 120–126.

[28] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM J. Computing* 26 (1997), 1484–1509.

[29] P. W. Shor, Fault-tolerant quantum computation, in *Proc. 37nd Annual Symposium on Foundations of Computer Science,* IEEE Computer Society Press, Los Alamitos, CA (1996), 56–65.

[30] D. R. Simon, On the power of quantum computation, *SIAM J. Computing* 26 (1997), 1474–1483.

[31] A. Steane, Multiple particle interference and quantum error correction, *Proc. Roy. Soc. London Ser. A* 452 (1996), 2551–2577.

[32] A. Yao, Quantum circuit complexity, in *Proceedings of the 34th Annual Symposium on Foundations of Computer Science,* IEEE Computer Society Press, Los Alamitos, CA (1993), 352–361.

[33] C. Zalka, Efficient simulation of quantum systems by quantum computers, *Proc. Roy. Soc. London Ser. A* 454 (1998), 313–322.

[34] C. Zalka, Fast versions of Shor's quantum factoring algorithm, LANL e-print quant-ph/9806084 (1998), available online at `http://xxx.lanl.gov/`.

[35] W. H. Zurek, Decoherence and the transition from quantum to classical, *Physics Today* 44 (1991) 36–44.

Peter W. Shor
AT&T Labs – Research
180 Park Ave.
Florham Park, NJ 07932 USA
shor@research.att.com