DOI:10.1145/1646353.1646375

What quantum algorithms outperform classical computation and how do they do it?

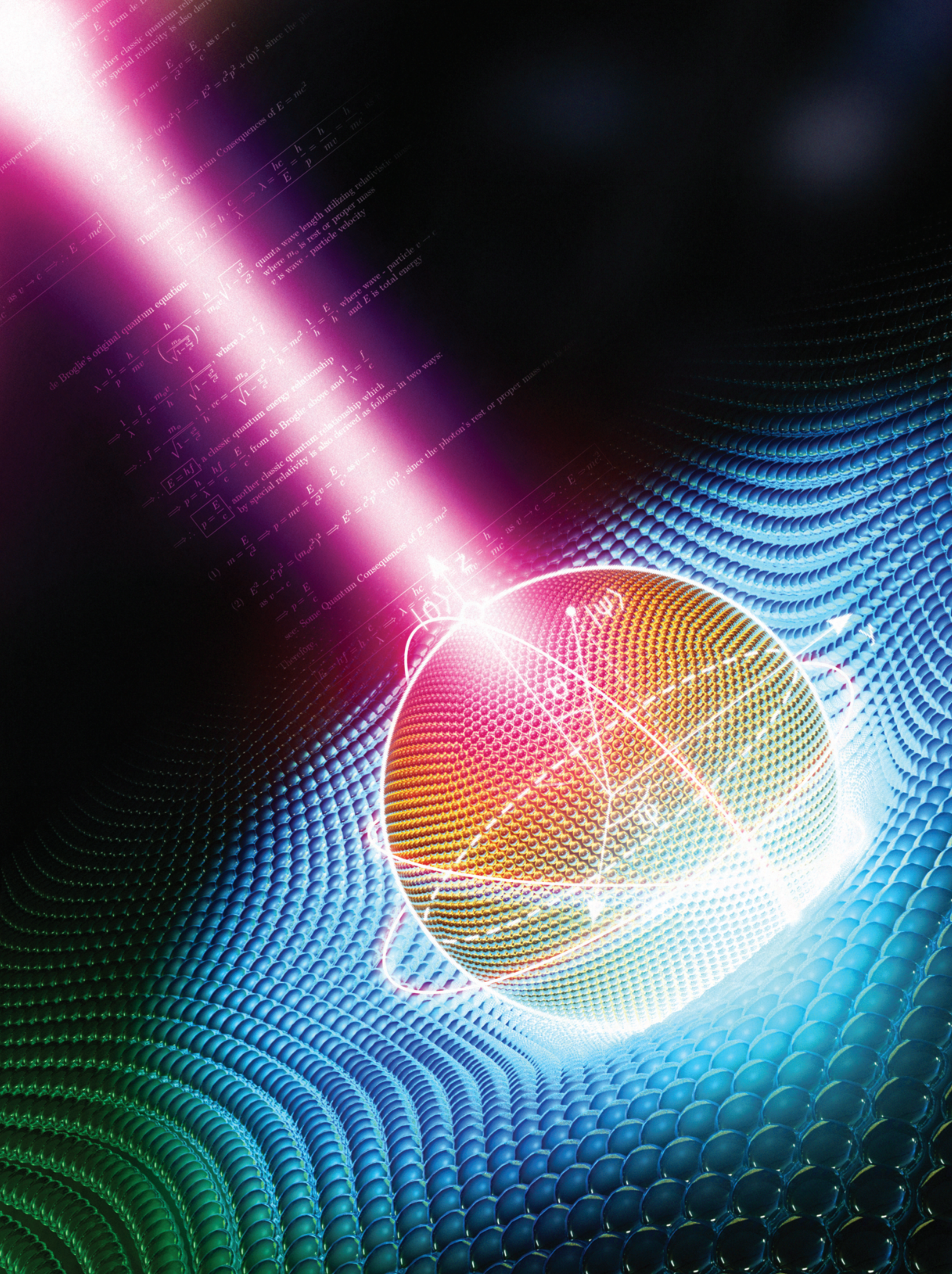
BY DAVE BACON AND WIM VAN DAM

Recent Progress in Quantum Algorithms

IT IS IMPOSSIBLE to imagine today's technological world without algorithms: sorting, searching, calculating, and simulating are being used everywhere to make our everyday lives better. But what are the benefits of the more philosophical endeavor of studying the notion of an algorithm through the perspective of the physical laws of the universe? This simple idea, that we desire an understanding of the algorithm based upon physics seems, upon first reflection, to be nothing more than mere plumbing in the basement of computer science. That is, until one realizes that the pipes of the universe do not seem to behave like the standard components out of which we build a computer, but instead obey the counterintuitive laws of quantum theory. And, even more astoundingly, when one puts these quantum parts together, one gets a notion of the algorithm—the quantum algorithm—whose computational power appears to be fundamentally more efficient at carrying out certain tasks than algorithms written for today's, nonquantum, computers. Could this possibly be true: that there is a more fundamental notion of algorithmic efficiency for computers built from quantum components? And, if this is true, what exactly is the power of these quantum algorithms?

The shot that rang round the computational world announcing the arrival of the quantum algorithm was the 1994 discovery by Peter Shor that quantum computers could efficiently factor natural numbers and compute discrete logarithms.[24] The problem of finding efficient algorithms for factoring has been burning the brains of mathematicians at least as far back as Gauss who commented upon the problem that "the dignity of science seems to demand that every aid to the solution of such an elegant and celebrated problem be zealously cultivated." Even more important than the fact that such a simple and central problem has eluded an efficient algorithmic solution is that the lack of such an efficient algorithm has been used as a justification for the security of public key cryptosystems, like RSA encryption.[23] Shor's algorithm, then, didn't just solve a problem of pure academic interest, but instead ended up showing how quantum computers could break the vast majority of cryptographic protocols in widespread use today. If we want the content of our publicly key encrypted messages to remain secret not only now, but also in the future, then Shor's algorithm redefines the scope of our confidence in computer security: we communicate securely, today, given that we cannot build a large scale quantum computer tomorrow.

Given the encryption breaking powers promised by quantum computers, it was natural that, in the decade following Shor's discovery, research has focused largely on whether a



quantum computer could be built. While there currently appear to be no fundamental obstacles toward building a large scale quantum computer (and even more importantly, a result known as the "threshold theorem"[1, 16–18, 25] shows that quantum computers can be made resilient against small amounts of noise, thereby confirming that these are not analog machines), the engineering challenges posed to build an RSA breaking quantum computer are severe and the largest quantum computers built to date have less than 10 quantum bits (qubits).[13, 19] But regardless of the progress in building a quantum computer, if we are to seriously consider our understanding of computation as being based upon experimental evidence, we will have to investigate the power of quantum algorithms. Christos Papadimitriou said in a recent interview[26] that the theory of computational complexity is such a difficult field because it is nearly impossible to prove what everyone knows from experience. How, then, can we even begin to gain an understanding of the power of quantum computers if we don't have one from which to gain such an experience? Further, and perhaps even more challenging, quantum algorithms seem to be exploiting the very effects that make quantum theory so uniquely counterintuitive.[6] Designing algorithms for a quantum computer is like building a car without having a road or gas to take it for a test drive.

In spite of these difficulties, a group of intrepid multidisciplinary researchers have been tackling the question of the power of quantum algorithms in the decades since Shor's discoveries. Here we review recent progress on the upper bounding side of this problem: what new quantum algorithms have been discovered that outperform classical algorithms and what can we learn from these discoveries? Indeed, while Shor's factoring algorithm is a tough act to follow, significant progress in quantum algorithms has been achieved. We concentrate on reviewing the more recent progress on this problem, skipping the discussion of early (but still important) quantum algorithms such as Grover's algorithm[12] for searching (a quantum algorithm that can search an unstructured space quadratically faster than the best classical algorithm), but explaining some older algorithms in order to set context. For a good reference for learning about such early, now "classic" algorithms (like Grover's algorithm and Shor's algorithm) we refer the reader to the textbook by Nielsen and Chuang.[21] Our discussion is largely ahistoric and motivated by attempting to give the reader intuition as to what motivated these new quantum algorithms. Astonishingly, we will see that progress in quantum algorithms has brought into the algorithmic fold basic ideas that have long been foundational in physics: interference, scattering, and group representation theory. Today's quantum algorithm designers plunder ideas from physics, mathematics, and chemistry, weld them with the tried and true methods of classical computer science, in order to build a new generation of quantum contraptions which can outperform their classical counterparts.

Quantum Theory in a Nutshell

Quantum theory has acquired a reputation as an impenetrable theory accessible only after acquiring a significant theoretical physics background. One of the lessons of quantum computing is that this is not necessarily true: quantum computing can be learned without mastering vast amounts of physics, but instead by learning a few simple differences between quantum and classical information. Before discussing quantum algorithms we first give a brief overview of why this is true and point out the distinguishing features that separate quantum information from classical information.

To describe a deterministic n-bit system it is sufficient to write down its configuration, which is simply a binary string of length n. If, however, we have n-bits that can change according to probabilistic rules (we allow randomness into how we manipulate these bits), we will instead have to specify the probability distribution of the n-bits. This means to *specify* the system we require 2^n positive real numbers describing the probability of the system being in a given configuration. These 2^n numbers must sum to unity since they are, after all, probabilities. When we observe a classical system, we will always find it to exist in one particular configuration (i.e. one particular binary string) with the probability given by the 2^n numbers in our probability distribution.

Now let's turn this approach to quantum systems, and consider a system made up of n qubits. Again, n qubits will have a configuration which is just a length n binary string. When you observe n qubits you will only see an n bit configuration (thus when you hear someone say that a qubit is both zero and one at the same time you can rely on your common sense tell them that this is absurd). But now, instead of describing our system by 2^n probabilities, we describe a quantum system by 2^n *amplitudes*. Amplitudes, unlike probabilities (which were positive real numbers and which summed to unity), are complex numbers which, when you take their absolute value-squared and add them up, sum to unity. Given the 2^n amplitudes describing a quantum system, if you observe the system, you will see a particular configuration with a probability given by the modulus squared of the amplitude for that configuration. In other words, quantum systems are described by a set of 2^n complex numbers that are a bit like square roots of probabilities (see Figure 1).

So far we have just said that there is this different description for quantum systems, you describe them by amplitudes and not by probabilities. But does this really have a consequence? After all the amplitudes aren't used so far, except to calculate probabilities. In order to see that yes, indeed, it does

Figure 1. Classical versus quantum information.

On the left, the classical bit is described by two nonnegative real numbers for its probabilities Pr(0) = 1/3 and Pr(1) = 2/3. The quantum bit on the right, instead, has two complex valued amplitudes that give the (same) probabilities by taking the absolute value-squared of its entries. When a quantum system has such a description with nonzero amplitudes, one says that the system is in a superposition of the 0 and 1 configurations.

$$\text{Classical bit: } \begin{bmatrix} \frac{1}{3} \\[2ex] \frac{2}{3} \end{bmatrix} \qquad \text{Quantum bit: } \begin{bmatrix} \frac{-1}{\sqrt{3}} \\[2ex] \frac{1+i}{\sqrt{3}} \end{bmatrix}$$

have a profound consequence, we must next describe how to update our description of a system as it changes in time. One can think about this as analyzing an *algorithm* where information in our computing device changes with time according to a set of specific recipe of changes.

For a classical probabilistic computing device we can describe how it changes in time by describing the conditional probability that the system changed into a new configuration given that it was in an old configuration. Such a set of conditional probabilities means that we can describe a probabilistic computing action by a stochastic matrix (a matrix whose entries are positive and whose columns sum to unity). A classical probabilistic algorithm can then be viewed as just a set of stochastic matrices describing how probabilities propagate through the computing device. If the classical probabilistic algorithm starts with n bits and ends with m bits, then the stochastic matrix describing the algorithm will be a 2^m by 2^n matrix.

What is the analogous procedure for a quantum system? Well instead of specifying conditional probabilities of a new configuration given an old configuration, in a quantum system you need to specify the conditional amplitude of a new configuration given an old configuration. In the quantum world, the matrix of conditional amplitudes has two major differences from the classical probabilistic setting. The first is that quantum systems evolve reversibly and thus the matrix is 2^n by 2^n (corresponding to the amplitude of every configuration to change into any other configuration). The second is that, in order to preserve the sum of the squares of those amplitudes, which should be 1 throughout, this matrix is a unitary matrix, meaning the entries of the matrix are complex numbers, and that the rows (and columns) of this matrix are orthonormal. Thus a quantum algorithm for a quantum system is given by a unitary matrix of conditional amplitudes.

What consequence does this change from probabilities to amplitudes and from stochastic matrices to unitary matrices have for the notion of an algorithm? This is, of course, the essential question at hand when considering quantum algorithms. In this survey we single out three major differences—quantum interference, the deep relationship between symmetries and quantum mechanics, and quantum entanglement—and show how they are related to recent progress in quantum algorithms.

Interference and the Quantum Drunkard's Walk

The first of our claimed differences between quantum computers and classical computers was that the former led to effects of quantum interference. What is interference and how can it lead to new efficient algorithms?

To illustrate the ideas of interference, consider a random walk on a line. The standard, classical drunkard's walk on a line refers to situation where the walker is allowed to step either forward or backward with equal probability every unit time step. When starting at position 0 at time zero, then after one time step there is an equal probability to be at locations +1 and –1. After the next time step, there is a one-fourth probability of being at positions –2 and 2 and one half probability of being at position 0. Notice here that the probability of reaching zero was the sum of two probabilities: the probability that the drunkard got to 0 via 1 and the probability that it got to 0 via –1. Random walks on structures more complicated than a line are a well-known tool in classical algorithms.

Suppose that we want to construct a quantum version of this drunkard's walk. To specify a quantum walk, we need, instead of a probability for taking a step forward or backward, an amplitude for doing this. However we also need to make sure that the unitary nature of quantum theory is respected. For example, you might think that the quantum analogy of a classical walk is to take a step forward and a step backward with amplitude one over the square root of two (since squaring this gives a probability of one half). If we start at 0, then after one step this prescription works: we have equal amplitude of one over square root of two of being at either 1 or –1. If we measure the walker after this first step, the probability of being at 1 or –1 is both one half. But if we run this for another time step, we see that we have an amplitude of ½ to be at –2 or 2 and an amplitude 1 to be at 0. Unfortunately if we square these numbers and add them up, we get a number greater than unity, indicating that the evolution we have described is not unitary.

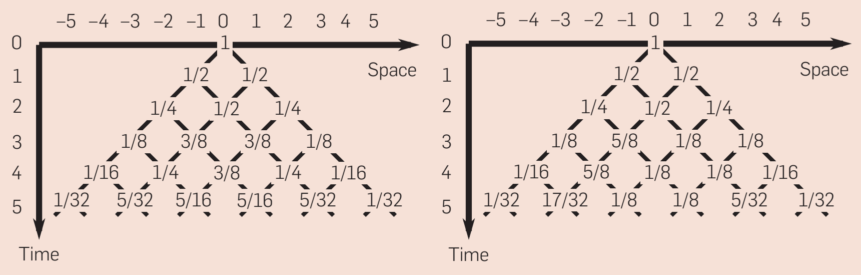
The solution to this problem is to let the drunkard flip a quantum coin at each time step, after which he steps in the direction indicated by the quantum coin. What is a quantum coin? A quantum coin is simply a qubit whose two configurations we can call "forward" and "backward" indicating the direction we are supposed to move after flipping the quantum coin. How do we flip such a coin? We apply a unitary transform. This unitary transform must specify four amplitudes. One choice of such a unitary transform that seems to mimic the drunkard's walk is to assign all conditional amplitudes a value of one over the square root of two, with the exception of the amplitude to change from the configuration "forward" to the configuration "backward," which, due to unitarity, we assign the amplitude negative one over square root of two. In other words the unitary transform we apply to flip the coin is specified by the transition matrix

$$H = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}. \tag{1}$$

If we follow this prescription for a quantum random walk with the drunkard initially positioned at zero, one quickly sees that something strange happens. Consider, for instance, the probability distribution formed by the quantum walk had we measured the walker's position after three time steps (see Figure 2). Then the probability of getting to +1 for the drunkard is ⅛. For a classical walk the similar number would be ⅜. What is going on here? Well if you trace back how he could have gotten to +1 in three steps, you'll see that there are three paths it could have used to get to this position. In the classical world each of these is traversed with equal probability, adding a contribution of ⅛ for each step.

Figure 2. Classical (top) and quantum (bottom) random walks.

The probability of reaching a particular point in space and time, given that we measure the position at that time, is listed on the vertices.



But in the quantum world, two of these paths contribute equal but oppositely to the *amplitude* to get to this position. In other words these two paths interfere with each other. Because amplitudes, unlike probabilities, don't have to be positive numbers, they can add up in ways that cancel out. This is the effect known as quantum interference. It is the same interference idea which you see when two water waves collide with each other. But note an important difference here: amplitudes squared are probabilities. In water waves, the heights interfere, not anything related to the probabilities of the waves. This is the peculiar effect of quantum interference.

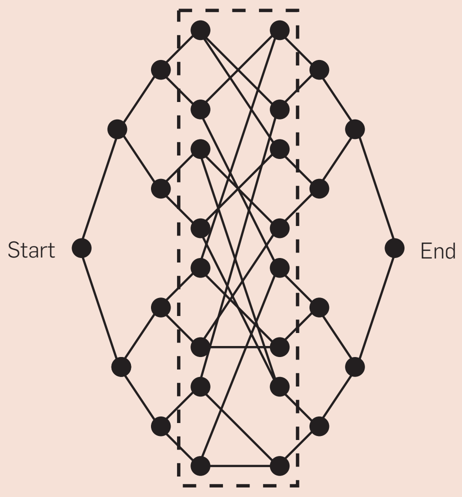
Quantum random walks were actually first described by physicists in 1993,[2] but only with the rise of interest in quantum computers was it asked whether these walks could be used as a computational tool. An alternative, continuous time version of these algorithms (tacking more closely to ideas in physics) has also been developed by Farhi and Gutmann.[9] Given these quantum random walks, a natural question is what does this have to do with algorithms? Well, the first observation is that quantum random walks behave in strange ways. For instance a well-known property of classical random walks on a line is that the expected standard deviation of a random walk as a function of the number of steps taken, T, scales like the square root of T. However, for a quantum random walk the standard deviation can actually spread linearly with T. Remarkably, this difference has been well known to physicists for a long time: it turns out that the quantum random walk defined above is closely related to the Dirac equation for a one-dimensional electron (the Dirac equation is a way to get quantum mechanics to play nicely with the special theory of relativity, and is a basic equation used in modern quantum field theory). This discovery that quantum algorithms seem to explore space quadratically faster than classical random walks has recently been shown to lead to quantum algorithms that polynomially outperform their classical cousins.

One example of an algorithm based upon quantum random walks is the algorithm for element distinctness due to Ambainis.[3] The element distinctness problem is, given a function f from $\{1, 2, \ldots, N\}$ to $\{1, 2, \ldots, N\}$ determine whether there exists two indices $i \neq j$ such that $f(i) = f(j)$. Classically this requires $\Omega(N)$ queries to the function f. Ambainis showed how a quantum random walk algorithm for this problem could be made to work using $O(N^{2/3})$ queries: an improvement which has not been achieved using any other quantum methods to date. Other algorithms that admit speedups of a similar nature by using quantum random walks are spatial search (searching a spatially d-dimensional space),[4] triangle finding,[20] and verifying matrix products.[7] Quantum random walks algorithms, then, are a powerful tool for deriving new quantum algorithms.

These examples all achieved polynomial speedups over the best possible classical algorithms. Given that quantum random walks can be used to polynomially outperform classical computers at some tasks, a natural question is whether quantum computers can be used to exponentially outperform classical computers. The answer to this question was first given by Childs et al.,[8] who showed that a quantum random walk could traverse a graph exponentially faster than any possible classical algorithm walking on this graph. In Figure 3 we show the graph in question: the crux of the idea is that a quantum algorithm, by constructively or destructively interfering, can traverse this graph, while a classical algorithm will always get stuck in the middle of the graph. Construc-

Figure 3. An example of a graph arising in the quantum random walk problem considered by Childs et al.[8]

In this problem one is given access to a function that takes as input a vertex and returns a list of the vertex's neighbors. The goal of the problem considered by Childs et al. is, by querying the function as few times as possible, traverse from the start vertex to the end vertex. The graphs considered are two full binary trees pasted together with a random cycle (in the example, the cycle resides inside the dashed box) whose roots are the start and end vertices. The quantum algorithm starts at the start vertex and then performs a quantum diffusion to the end vertex. The random cycle in the middle does not destroy this diffusion, since all paths contribute equally to this diffusion. For a graph of depth d, the quantum walk will find the end vertex by querying the local vertex function a polynomial number of times in d. The best classical algorithm can be shown to require querying the function for local vertex information exponentially many times in d.



tive interference refers to the condition where quantum evolution causes amplitudes to increase in absolute magnitude (and hence in probability) while destructive interference refers to where the evolution causes amplitudes to decrease in absolute magnitude (and hence decrease in probability). In spite of this success, the above problem, traversing this graph, does not appear to have a good algorithmic use. Thus a subject of great research interest today is whether there are quantum random walk algorithms that offer exponential speedups over classical algorithms for interesting algorithmic problems.

Quantum Algorithms and Game Playing

Quantum interference, the ability of multiple computational paths to add or detract amplitudes and thus lower and raise probabilities, is an effect well known to physicists. Given this, it is interesting to ask whether other techniques from physicists toolbox might also be of use in algorithms. A great example of this approach was the recent discovery by Farhi et al.[10] of a quantum algorithm that outperforms all possible classical algorithms for the evaluation of NAND tree circuits. This algorithm was derived, amazingly, by considering the scattering of wave packets off certain binary trees. As a quintessential physics experiment involves shooting one quantum system at another and observing the resulting scattered 'outputs,' physicists have developed a host of tools for analyzing such scattering experiments. It was this approach that led the above authors to the following important new quantum algorithm.

To illustrate the NAND tree problem consider the following two player game. The players are presented with a complete binary tree of depth k. On the leaves of the tree are labels that declare whether player A or player B wins by getting to this node. At the beginning of a match, a marker is placed at the root of the tree. Players take alternating turns moving this marker down a level in the tree, choosing one of the two possible paths, with the goal, of course, of ending up at a leaf labeled by the player's name. A natural question to ask is if it is always possible for player A, with its first move, to win the game. Evaluating whether this is the case can be deduced inductively in the following way. Suppose player A makes the last move. Then player A will be able to win if the marker is on a node with at least one of its children labeled "A wins" hence we should label such internal nodes with "A wins" as well. This line of reasoning holds in general for all internal nodes on which A makes a move: as soon as one of its children has the label "A wins," then the node inherits the same conclusion. On the other hand, if none of the children has this label, then we can conclude that "B wins." Player B will, of course, be reasoning in a similar manner. Thus we can see that player A will win, starting from a node of height two, only if both of the children of the node lead to positions where A wins. We can then proceed inductively using this logic to evaluate whether player A can always win the game with a move originating from the root of the tree. If we label the leaves where player A wins by 1 and where player B wins by 0, then we can compute the value of the root node (indicating whether player A can always win) by representing the interior layers of the tree by alternating layers of AND and OR gates. Further it is easy to see that one can transform this from alternating layers of AND and OR gates to uniform layers of NAND (negated AND) gates, with a possible flipping of the binary values assigned to the leaves.

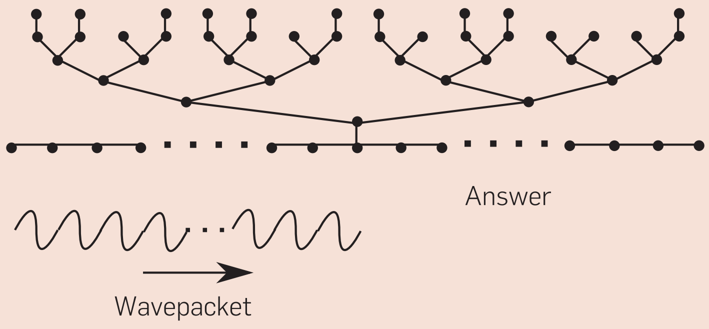
We have just shown that the problem of evaluating whether the first player has a series of moves that guarantees victory is equivalent to evaluating the value of a NAND tree circuit given a labeling the leaves of the tree. Further, if the player can evaluate any interior value of the NAND tree, then one can then use this to actually win the game. If such a procedure is available one can simply use the algorithm to evaluate the two trees and if one of them is always a win, take that move. Thus the problem of evaluating the value of the NAND tree is of central importance for winning this game. The NAND tree is an example of the more general concept of a game tree which is useful for study of many games such as Chess and Go. In these later games, more than two moves are available, but a similar logic for evaluating whether there is a winning strategy applies. This problem, of which the NAND tree circuit is the smallest example, is a central object in the study of combinatorial games.

One can now ask: how costly is it to evaluate the NAND tree: how many nodes does one need to query in order to compute the value of the NAND tree? One could evaluate every leaf and compute the root, but certainly this is wasteful: if you ever encounter a subtree which evaluates to 0, you know that the parent of this subtree must evaluate to 1. A probabilistic recursive algorithm is then easy to think up: evaluate a subtree by first evaluating randomly either its left or right subtree. If this (left or right) subtree is 0, then the original subtree must have value 1. If not, evaluate the other subtree. This method, known as alpha–beta pruning, has a long history in artificial intelligence research. For the NAND tree, one can show that by evaluating about $\Omega(N^{0.753})$ of the N leaves one can calculate the value of the NAND tree with high probability. It is also known that this value for the number of leaves needed to be queried is optimal.

For a long period of time it was uncertain whether quantum computers could perform better than this. Using standard lower bounding methods, the best lower bound which could be proved was a $O(N^{1/2})$, yet no quantum algorithm was able to achieve such a speedup over the best classical algorithm. Enter onto the scene the physicists Farhi, Goldstone, and Gutmann. These authors considered a continuous quantum random walk of a strange form. They considered a quantum random walk on the graph formed by a binary tree (of size related to the NAND tree being evaluated) attached to a long runway (see Figure 4). They then showed how, if one constructed an initial quantum system whose initial state was that of a quantum system moving to the right towards the binary tree, one could then obtain the value of the NAND tree by seeing whether such a quantum system scattered back off the binary tree, or passed through along the other side of the runway. The time required to see this scattering or lack of scattering was shown to be proportional to $O(N^{1/2})$. In other words, the NAND tree could be evaluated by using

Figure 4. The NAND tree algorithm of Farhi, Goldstone, and Gutmann.[10]

First, a tree is constructed where the presence or absence of leaves at the top of the tree corresponds to the binary input values to the NAND tree problem. Next, a wavepacket is then constructed which, if the tree were not attached, would propagate to the right. When the tree is attached, as shown, the value of the NAND tree can be determined by running the appropriate quantum walk and observing whether the wave packet passes to the right of the attached tree or is reflected backwards.



$O(N^{1/2})$ time by scattering a wave packet off of a binary tree representing the NAND tree problem. A few simple modifications can bring this in line with the standard computer scientists definition of a query algorithm for the NAND tree problem. Presto, out of a scattering experiment, one can derive a quantum algorithm for the NAND tree problem which gives a $O(N^{1/2})$ algorithm outperforming a classical computer science algorithm. Building upon this work, a variety of different trees with different branching ratios and degrees of being balanced have been explored showing quantum speedups. Indeed one remarkable aspect of much of this work is that while in many cases the classical versions of these problems do not have matching upper and lower bounds, in the quantum case matching upper and lower bounds can now be achieved.

Finding Hidden Symmetries

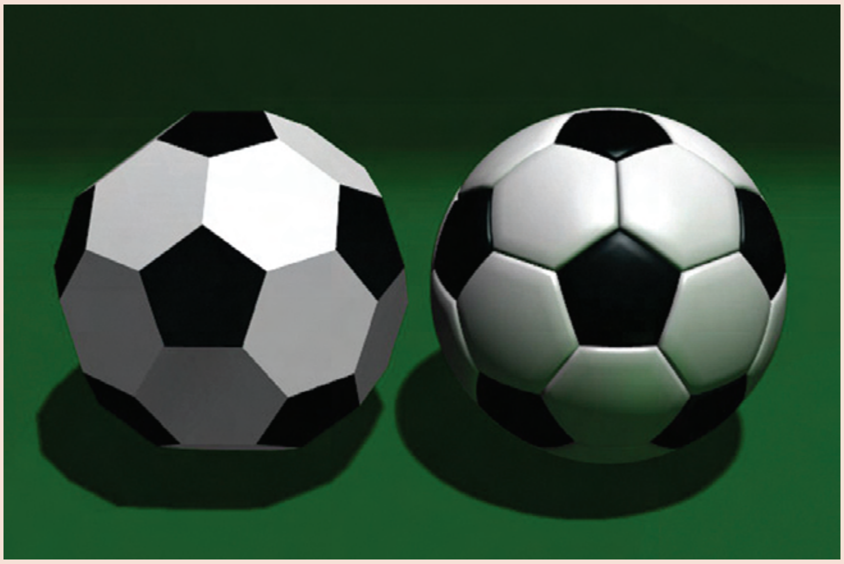
If interference is a quantum effect that leads to polynomial speedups, what about the quantum algorithms that appear to offer exponential speedups, like in Shor's algorithm for factoring or the quantum random walk algorithm of Childs et al. described here? Here it seems that just using interference by itself is not sufficient for gaining such extraordinary power. Instead, in the vast majority of cases where we have exponential speedups for quantum algorithms, a different candidate emerges for giving quantum computers power: the ability to efficiently find hidden symmetries. Here we review recent progress in algorithms concerning hidden symmetries. In many respects these algorithms date back to the earliest quantum algorithms, a connection we first briefly review, before turning to more modern ways in which this has influenced finding new quantum algorithms.

We say an object has *symmetry* if "we can do something to it without changing it." The things we can do are described by the elements of a group and the object itself is a function that is defined on the same group. That this does not have to be as abstract as it seems is illustrated in Figure 5 for the group of three-dimensional rotations and the icosahedral symmetry of a soccer ball.

Given a group G the symmetry of a function f defined on G can range from the trivial (when only the identity of G leaves f unchanged) to the maximum possible symmetry where f remains unchanged under all possible group operations. The most interesting cases happen when f is invariant under only a proper *subgroup* H of G and the task of finding this H, given f, is known as the *hidden subgroup problem*. For many different types of groups we know how to solve this problem efficiently on a quantum computer, while no classical algorithm can perform the same feat. We claim that this is because quantum computers can more efficiently exploit problems with hidden symmetries.

To illustrate how quantum computers are better suited to deal with symmetries, let's talk about the simplest symmetry one can talk about: the symmetry of flipping a bit. Consider the operation X of negating a bit and the identity operation I. If we perform X twice, we obtain the operation I of doing nothing at all, which shows that I and X together form a group. Next, consider representing how I and X

Figure 5. The symmetries of a soccer ball.



Of all the possible three-dimensional rotations that one can apply, only a finite number of them leave the image of a standard soccer ball unchanged. This subgroup, the icosahedral rotation group with its 60 elements, therefore describes the symmetries of the object; http://en.wikipedia.org/wiki/File: Trunc-icosa.jpg/

operate on a classical probabilistic bit. Such a binary system is described by a two-dimensional vector of probabilities, corresponding to the probability p_0 of being in 0 and p_1 of being in 1. The operations I and X can then be represented on this system as the two-by-two matrices

$$I \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \text{ and } X \equiv \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

In group theoretic parlance, we say that these two matrices form a *representation* of the group, which effectively means that the multiplication among these matrices mimics the operation among the elements of the group that is being represented.

But now notice how the matrices for I and X act on the vector space $\mathbb{R}^2$. Naturally, the identity matrix I leaves all vectors unchanged, but the X matrix acts in a more interesting way. If X acts on the symmetric vector $[1, 1]$, then, like I, it preserves this vector. If, on the other hand, X operates upon the vector $[1, -1]$, then it multiplies this vector by -1. This new vector $[-1, 1]$ still sits in the one-dimensional subspace spanned by the original $[1, -1]$, but the direction of the vector has been reversed. In other words, the act of flipping a bit can naturally be represented down into its action upon two one-dimensional subspaces: on the first of these the group always acts trivially, while on the other it always acts by multiplying by the scalar -1. Now we can see why classical probabilistic information is at odds with this symmetry: while we can create a symmetric probability distribution $[\frac{1}{2}, \frac{1}{2}]$ wherein the bit flip X preserves this distribution, we cannot create the other probability distribution that transforms according to the multiplication by -1: doing so would require that we have negative probabilities. But wait, this is exactly what the amplitudes of quantum computers allow you to do: to prepare and analyze quantum information in all the relevant subspaces associated with group operations such as flipping a bit. Unlike classical computers, quantum computers can analyze symmetries by realizing the unitary transforms which directly show the effects of these symmetries. This, in a nutshell, is why quantum algorithms are better adapted to solve problems that involve symmetries.

The idea that symmetry is the excelsior of exponential quantum speedups now has considerable evidence in its favor and is one of the major motivators for current research in quantum algorithms. Shor's algorithm for factoring works by converting the problem of finding divisors to that of finding periods of a function defined over the integers, which in turn is the problem of determining the translational symmetries of this function. In particular Shor's algorithm works by finding the period of the function $f(x) = r^x \bmod N$ where r is a random number coprime with N, the number one wishes to factor. If one finds the period of this function, i.e. the smallest nonzero p such that $f(x) = f(x + p)$, then one has identified a p such that $x^p = 1 \bmod N$. If p is even (which happens with constant probability for random x), then we can express this equation as $(x^{p/2} + 1)(x^{p/2} - 1) = 0 \bmod N$. This implies that the greatest common divisor of $x^{p/2} + 1$ and N or the greatest common divisor of $x^{p/2} - 1$ and N is a divisor of N. One can then use the Euclidean algorithm to find a factor of N (should it exist). Thus one can efficiently factor assuming one can find the period p of $f(x)$. This fact was known before Shor's discovery; the task of determining the period p is what requires a quantum computer.

How then, can a quantum algorithm find the period p of a function f? The answer is: by exploiting the just described friendly relationship between quantum mechanics and group theory. One starts with a system of two quantum registers, call them left and right. These are prepared into a state where with equal amplitude the left register contains a value x and the right register carries the corresponding function value $f(x)$. The hidden symmetry of this state is captured by the fact that it remains unchanged if we would and p (or a multiple of p) to the left register; adding a nonmultiple of p will, on the other hand, change the state. To extract this hidden symmetry, let us view the amplitudes of the state as the values of a function from n bit strings to the complex numbers. We would like to use a quantum version of the Fourier transform to extract the symmetry hidden in this function. Why the Fourier transform? The answer to this is that the Fourier transform is intimately related to the symmetry of addition modulo N. In particular if we examine the process of addition where we have performed a Fourier transform before the addition and an inverse Fourier transform after the addition, we will find that it is now transformed from an addition into multiplication by a phase (a complex number z such that $|z| = 1$). Addition can be represented on a quantum computer as a permutation matrix: a matrix with only a single one per column and row of the matrix. If we examine how such a matrix looks in the basis change given by the Fourier transform, then we see that this matrix only has entries on the diagonal of the matrix. Thus the Fourier transform is exactly the unitary transform which one can use to "diagonalize the addition matrix" with respect to the symmetry of addition, which in turn is exactly the form of the symmetry needed for period finding.

The output of the quantum Fourier transformation will reveal to us which symmetries the state has, and by repeating this Fourier sampling a few times we will be able to learn the exact subgroup that the state hides, thus giving us the period p (and hence allowing us to factor). Crucially the quantum Fourier transform can be implemented on a number of qubits logarithmic in the size of the addition group, $\log N$, and in a time polynomial in $\log N$ as well. If one were to attempt to mimic Shor's algorithm on a classical computer, one would need to perform a Fourier transform on N classical pieces of data, which would require $N \log N$ time (using the fast Fourier transform). In contrast, because Shor's quantum algorithm acts on quantum amplitudes, instead of on classical configuration data, it leads to an efficient quantum algorithm for factoring.

This symmetry analysis results from the basics of the theory of group representation theory: symmetries are described by groups, and the elements of these groups can be represented by unitary matrices. This is something that classical probabilistic computers cannot exploit: the only way to represent a group on a classical computer is to represent it as by deterministic permutation. But while a group can be represented by unitary matrices, no such representation is possible using stochastic matrices. This, at its heart, appears to be one of the key reasons that quantum computers offer exponential benefits for some problems over classical computers.

Given that Shor's algorithm exploits symmetry in such a successful way, it is natural to search for other problems that involve hidden symmetries. Following Shor's discovery it was quickly realized that almost all prior quantum algorithms could be cast in a unifying form as solving the hidden subgroup problem for one group or the other. For Shor's algorithm the relevant group is the group of addition modulo N. For the discrete logarithm problem the relevant group is the direct product of the groups of addition modulo N. Indeed it was soon discovered that for all finite Abelian groups (Abelian groups are those whose elements all commute with each other) quantum computers could efficiently solve the hidden subgroup problem. A natural follow-up question is: what about the non-Abelian hidden subgroup problem? And, even more importantly, would such an algorithm be useful for any natural problems, as the Abelian hidden subgroup problem is useful for factoring?

One of the remarkable facts about the problem of factoring is its intermediate computational complexity. Indeed, if one examines the decision version of the factoring problem, one finds that this is a problem which is in the complexity class NP and in the complexity class Co-NP. Because of this fact it is thought to be highly unlikely that it is NP-complete, since if it were, then the polynomial hierarchy would collapse in a way thought unlikely by complexity theorists. On the other hand, there is no known classical algorithm for factoring. Thus factoring appears to be of Goldilock's complexity: not so hard as to revolutionize our notion of tractability by being NP-complete, but not so easy as to admit efficient classical solution. There are, surprisingly, only a few problems which appear to fit into this category. Among them, however, are the problems of graph isomorphism and certain shortest-vector in a lattice problems. Might quantum computers help at solving these problems efficiently?

Soon after Shor's algorithm was phrased as a hidden subgroup problem, it was realized that if you could efficiently solve the hidden subgroup problem over the symmetric group (the group of permutations of n objects), then you would have an efficient quantum algorithm that solves the graph isomorphism problem. Further, Regev[22] showed how the hidden subgroup problem over the dihedral group (the group of symmetries of a regular polygon where one can not only rotate but also flip the object) relates to finding short vectors in a high dimensional lattice. Hence a hypothetical efficient quantum algorithm for this dihedral case could be used to solve such shortest vector problems. This in turn would break the public key cryptosystems that are based upon the hardness of these lattice problems, which are among the very few cryptosystems not broken by Shor's algorithm. As a result of these observations about the non-Abelian hidden subgroup problem, designing quantum algorithms for such groups has become an important part of the research in quantum computation. While a certain amount of progress has been achieved (by now we know of many non-Abelian groups over which the hidden subgroup problem can be solved efficiently), this problem remains one of the outstanding problems in the theory of quantum algorithms.

At the same time, going back to the Abelian groups, there has been quite some success in finding new applications of the quantum algorithm for the Abelian hidden subgroup problem, besides factoring and discrete logarithms. Hallgren[14] showed that there exists a quantum algorithm for solving Pell's equation (that is, finding integer solutions x, y to the cubic equation $x^2 - dy^2 = 1$, see Table 1), while Kedlaya[15] has described a quantum procedure that efficiently counts the number points of curves defined over finite fields. Furthermore, other efficient quantum algorithm has been found for, among other problems, determining the structure of black box groups, estimating Gauss sums, finding hidden shifts, and estimating known invariants.

Simulating Quantum Physics

A final area in which quantum algorithms have made progress goes back to the very roots of quantum computing and indeed of classical computing itself. From their earliest days, computers have been put to use in simulating physics. Among the difficulties that were

Table 1. Some examples of integer solutions (x, y) to Pell's equation $x^2 - dy^2 = 1$ for different values d.

Such solutions tell us what the units are of the number field Q[÷ $\sqrt{d}$] (the rational numbers extended with the irrational ÷ $\sqrt{d}$) and thereby solve the unit group problem. Hallgren's result shows how this problem can be solved efficiently on a quantum computer, while no such algorithm is known for classical computers.

d	x	y
2	3	2
3	2	1
5	9	4
⋮		
13	649	180
14	15	4
⋮		
6,009	1,316,340,106,327,253,158	1,698,114,661,157,803,451
	9,259,446,951,059,947,388	6,889,492,378,831,465,766
	4,013,975 ≈ 1.3×10^{44}	81,644 ≈ 1.6×10^{42}
6,013	40,929,908,599	527,831,340
⋮		

soon encountered in such simulations was that quantum systems appeared to be harder to simulate than their classical counterparts. But, of course, somehow nature, which obeys quantum theory, is already carrying out "the simulation" involved in quantum physics. So, if nature is carrying out the simulation, then should we be able to design a computer that also can perform this simulation? This was in fact the seed of the idea that led to the original notion of quantum computing by Feynman.[11]

To put this in perspective, consider the problem of simulating classical physics. The miracle of reproducing classical physics on a classical computer is that you can use many 'particles' with small state spaces (bits) to mimic a few particles that have very large state spaces. For this to be possible it is required that the number of bit configurations, $2^{(\text{number of bits})}$, is at least as big as the number of possible states of the physical system (which is the size of the particle's state space exponentiated with the number of particles). As a result, we can simulate the solar system on a laptop.

Quantum computing does the same thing for quantum mechanical systems; now $2^{(\text{number of qubits})}$ is the dimension of the state space and it allows us to simulate other quantum physical systems that consists of few particles with exponentially large state spaces. Here however, it appears essential that we rely on quantum computing components in order to simulate the truly quantum mechanical components of a physical system. A crucial question therefore is: which physical systems are interesting to simulate in such a manner?

While the complete answer to this question is not known, a deeper look at quantum algorithms for simulating quantum physics is now being undertaken in several places. As an example, a group of physical chemists have recently compared how useful quantum computers would be for computing the energy level structure of molecular systems.[5] This is a classical problem of physical chemistry, and our inability to perform these calculations robustly for large molecules is a bottleneck in a variety of chemical and biological applications. Could quantum computers help for solving this problem and outperforming the best classical algorithms? One of the exciting findings in studying this problem was that a small quantum computer, consisting of only a few hundred qubits, could already outperform the best classical algorithms for this problem. This small number makes it likely that among the first applications of a quantum computer will not be factoring numbers, but instead will be in simulating quantum physics. Indeed, we believe that a quantum computer will be able to efficiently simulate my possible physical system and that it therefore has the potential to have a huge impact on everything from drug discovery to the rapid development of new materials.

Conclusion

The discovery that quantum computers could efficiently factor is, even today, difficult to really appreciate. There are many ways to get out of the conundrum posed by this discovery, but all of these will require a fundamental rewriting of our understanding of either physics or computer science. One possibility is that quantum computers cannot be built because quantum theory does not really hold as a universal theory. Although disappointing for quantum computer scientists, such a conclusion would be a major discovery about one of the best tested physical theories—quantum theory. Perhaps there is a classical algorithm for efficiently factoring integers. This would be a major computer science discovery and would blow apart our modern public key cryptography. Or perhaps, just perhaps, quantum computers really are the true model of computing in our universe, and the rules of what is efficiently computable have changed. These are the dreams of quantum computer scientists looking for quantum algorithms on the quantum machines they have yet to be quantum programmed. C

References

1. Aharonov, D., Ben-Or, M. Fault-tolerant quantum computation with constant error rate. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing* (1997). ACM, 176–188.
2. Aharonov, Y., Davidovich, L., Zagury, N. Quantum random walks. *Phys. Rev. A 48*, 167 (1993).
3. Ambainis, A. Quantum walk algorithm for element distinctness. *SIAM J. Comput. 37* (2007), 210.
4. Ambainis, A., Kempe, J., Rivosh, A. Coins make quantum walks faster. In *Proceedings of the 16th Annual ACM SIAM Symposium on Discrete Algorithms* (2005), 1099.
5. Aspuru-Guzik, A., Dutoi, A., Love, P.J., Head-Gordon, M. Simulated quantum computation of molecular energies. *Science 309*, 5741 (2005).
6. Bell, J.S. On the Einstein Podolsky Rosen paradox. *Physics 1*, (1964), 195.
7. Buhrman, H. Špalek, R. Quantum verification of matrix products. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms* (2006), 880.
8. Childs, A.M., Cleve, R., Deotto, E., Farhi, E., Gutmann, S., Spielman, D.A. Exponential algorithmic speedup by quantum walk. In *Proceedings of the 35th ACM Symposium on Theory of Computing* (2003), 59–68.
9. Farhi, E., Gutmann, S. Quantum computation and decision trees. *Phys. Rev. A 58* (1998), 915.
10. Farhi, E., Goldstone, J., Gutmann, S. A quantum algorithm for the Hamiltonian NAND tree. Eprint arXiv:quant-ph/0702144, 2007.
11. Feynman, R. Simulating physics with computers. *Intl. J. Theor. Phys. 21* (1982), 467–488.
12. Grover, L. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computation* (New York, 1996). ACM, 212–219.
13. Häffner, H., Hänsel, W., Roos, C.F., Benhelm, J., al kar, D.C., Chwalla, M., Körber, T., Rapol, U.D., Riebe, M., Schmidt, P.O., Becher, C., Gühne, O., Dür, W., Blatt, R. Scalable multiparticle entanglement of trapped ions. *Nature 438* (2005), 643.
14. Hallgren, S. Polynomial-time quantum algorithms for pell's equation and the principal ideal problem. In *Proceedings of the 34th Annual ACM Symposium on the Theory of Computation* (New York, 2002). ACM, 653–658.
15. Kedlaya, K.S. Quantum computation of zeta functions of curves. *Comput. Complex. 15*, 1–19 (2006).
16. Kitaev, A. Quantum error correction with imperfect gates. In *Quantum Communication, Computing and Measurement* (New York, 1997). Plenum, 181–188.
17. Knill, E., Laflamme, R., Zurek, W.H. Resilent quantum computation. *Science 279* (1998), 342–345.
18. Knill, E., Laflamme, R., Zurek, W.H. Resilient quantum computation: error models and thresholds. *Proc. Roy. Soc. Lond. Ser. A 454* (1998), 365–384.
19. Leibfried, D., Knill, E., Seidelin, S., Britton, J., Blakestad, R.B., Chiaverini, J., Hume, D.B., Itano, W.M., Jost, J.D., Langer, C., Ozeri, R., Reichle, R., Wineland, D.J. Creation of a six-atom 'Schrödinger cat' state. *Nature 438* (2005), 639.
20. Magniez, F., Santha, M., Szegedy, M. Quantum algorithms for the triangle problem. In *Proceedings of the 16th Annual ACM SIAM Symposium on Discrete Algorithms* (2005), 1109.
21. Nielsen, M.A. Chuang, I.L. *Quantum Computation and Quantum Information*. Cambridge University Press, New York, 2000.
22. Regev, O. Quantum computation and lattice problems. In *43rd Symposium on Foundations of Computer Science* (IEEE Computer Society, 2002), 520–529.
23. Rivest, R.L., Shamir, A., Adleman, L. A method of obtaining digital signatures and public-key cryptosystems. *Commun. ACM 21* (1978), 120–126.
24. Shor, P.W. Algorithms for quantum computation: Discrete log and factoring. In *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science*. S. Goldwasser, ed. (Los Alamitos, CA, 1994). IEEE Computer Society, 124–134.
25. Shor, P.W. Fault tolerant quantum computation. In *Proceedings of the 37th Symposium on the Foundations of Computer Science* (Los Alamitos, CA, 1996), IEEE, 56–65.
26. Woehr, J. Online interview "A Conversation with Christos Papadimitriou". *Dr. Dobb's J.* July Dave Bacon is an assistant research professor in the Department of Computer Science and Engineering, Department of Physics, at the University of Washington, Seattle.

Dave Bacon (dabacon@cs.washington.edu) is an assistant research professor in the Department of Computer Science & Engineering, Department of Physics, University of Washington, Seattle, WA.

Wim van Dam (vandam@cs.ucsb.edu) is an associate professor in the Department of Computer Science, Department of Physics, University of California, Santa Barbara, Santa Barbara, CA.