Improved simulation of stabilizer circuits

Scott Aaronson[*]
Computer Science Department, University of California, Berkeley, California 94720, USA

Daniel Gottesman[†]
Perimeter Institute, Waterloo, Canada N2L 2Y5

(Received 25 June 2004; published 30 November 2004)

The Gottesman-Knill theorem says that a stabilizer circuit—that is, a quantum circuit consisting solely of controlled-NOT (CNOT), Hadamard, and phase gates—can be simulated efficiently on a classical computer. This paper improves that theorem in several directions. First, by removing the need for Gaussian elimination, we make the simulation algorithm much faster at the cost of a factor of 2 increase in the number of bits needed to represent a state. We have implemented the improved algorithm in a freely available program called CHP (CNOT-Hadamard-phase), which can handle thousands of qubits easily. Second, we show that the problem of simulating stabilizer circuits is complete for the classical complexity class $\oplus\mathsf{L}$, which means that stabilizer circuits are probably not even universal for *classical* computation. Third, we give efficient algorithms for computing the inner product between two stabilizer states, putting any n-qubit stabilizer circuit into a "canonical form" that requires at most $O(n^2/\log n)$ gates, and other useful tasks. Fourth, we extend our simulation algorithm to circuits acting on mixed states, circuits containing a limited number of nonstabilizer gates, and circuits acting on general tensor-product initial states but containing only a limited number of measurements.

DOI: 10.1103/PhysRevA.70.052328 PACS number(s): 03.67.Lx, 03.67.Pp, 02.70.−c

I. INTRODUCTION

Among the many difficulties that quantum computer architects face, one of them is almost intrinsic to the task at hand: how do you design and debug circuits that you cannot even simulate efficiently with existing tools¿ Obviously, if a quantum computer output the factors of a 3000-digit number, then you would not need to simulate it to verify its correctness, since multiplying is easier than factoring. But what if the quantum computer did not work? Ordinarily architects might debug a computer by adding test conditions, monitoring registers, halting at intermediate steps, and so on. But for a quantum computer, all of these standard techniques would probably entail measurements that destroy coherence. Besides, it would be nice to design and debug a quantum computer using classical computer-aided design (CAD) tools, *before* trying to implement it.

Quantum architecture is one motivation for studying classical algorithms to simulate and manipulate quantum circuits, but it is not the only motivation. Chemists and physicists have long needed to simulate quantum systems, and they have not had the patience to wait for a quantum computer to be built. Instead, they have developed limited techniques such as the quantum Monte Carlo (QMC) method [1] for computing properties of certain ground states. More recently, several general-purpose quantum computer simulators have appeared, including Oemer's quantum programming language QCL [2], the QUIDD (quantum information decision diagrams) package of Viamontes *et al.* [3,4], and the parallel quantum computer simulator of Obenland and Despain [5]. The drawback of such simulators, of course, is that their running time grows exponentially in the number of qubits. With a general-purpose package, then, simulating hundreds or thousands of qubits is out of the question.

A different direction of research has sought to find nontrivial classes of quantum circuits that *can* be simulated efficiently on a classical computer. For example, Vidal [6] showed that, so long as a quantum computer's state at every time step has polynomially bounded entanglement under a measure related to Schmidt rank, the computer can be simulated classically in polynomial time. Notably, in a follow-up paper [7], Vidal actually implemented his algorithm and used it to simulate one-dimensional quantum spin chains consisting of hundreds of spins. A second example is a result of Valiant [8], which reduces the problem of simulating a restricted class of quantum computers to that of computing the Pfaffian of a matrix. The latter is known to be solvable in classical polynomial time. However, Valiant's model has thus far not found any application, although Terhal and DiVincenzo have shown that it applies to a model of noninteracting fermions [9].

There is one class of quantum circuits that is known to be simulable in classical polynomial time, that does not impose any limit on entanglement, and that arises naturally in several applications. This is the class of *stabilizer circuits* introduced to analyze quantum error-correcting codes [10–13]. A stabilizer circuit is simply a quantum circuit in which every gate is a controlled-NOT (CNOT), Hadamard, phase, or one-qubit measurement gate (Fig. 1). We call a stabilizer circuit *unitary* if it does not contain measurement gates. Unitary stabilizer circuits are also known as Clifford group circuits.

Stabilizer circuits will almost certainly be used to perform the encoding and decoding steps for a quantum error-correcting code, and they play an important role in fault-

[*]Present address: Institute for Advanced Study, Princeton, NY 08540, USA. Electronic address: aaronson@ias.edu

[†]Electronic address: dgottesman@perimeterinstitute.ca

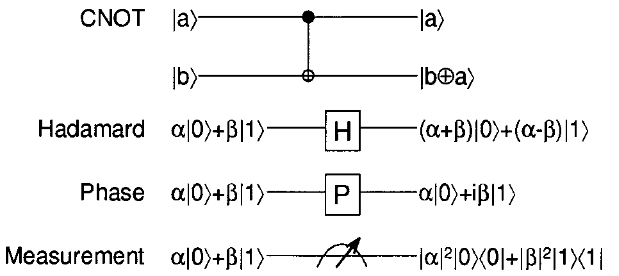

FIG. 1. The four types of gate allowed in the stabilizer formalism.

tolerant circuits. However, it was soon realized that the *stabilizer formalism* used to describe these circuits has many other applications. The stabilizer formalism is rich enough to encompass most of the “paradoxes” of quantum mechanics, including the Greenberger-Horne-Zeilinger (GHZ) experiment [14], dense quantum coding [15], and quantum teleportation [16]. On the other hand, it is not so rich as to preclude efficient simulation by a classical computer. That conclusion, sometimes known as the *Gottesman-Knill theorem*, is the starting point for the contributions of this paper.

Our results are as follows. In Sec. III we give a *tableau algorithm* for simulating stabilizer circuits that is faster than the algorithm directly implied by the Gottesman-Knill theorem. By removing the need for Gaussian elimination, this algorithm enables measurements to be simulated in $O(n^2)$ steps instead of $O(n^3)$ (where n is the number of qubits), at a cost of a factor of 2 increase in the number of bits needed to represent a quantum state.

Section IV describes CHP, a high-performance stabilizer circuit simulator that implements our tableau algorithm. We present the results of an experiment designed to test how CHP’s performance is affected by properties of the stabilizer circuit being simulated. CHP has already found application in simulations of quantum fault-tolerance circuits [39].

Section V proves that the problem of simulating stabilizer circuits is complete for the classical complexity class $\oplus\mathsf{L}$. Informally, this means that any stabilizer circuit can be simulated using CNOT gates alone; the availability of Hadamard and phase gates provides at most a polynomial advantage. This result removes some of the mystery about the Gottesman-Knill theorem by showing that stabilizer circuits are unlikely to be capable even of universal classical computation.

In Sec. VI we prove a *canonical form theorem* that we expect will have many applications to the study of stabilizer circuits. The theorem says that given any stabilizer circuit, there exists an equivalent stabilizer circuit that applies a round of Hadamard gates, followed by a round of phase gates, followed by a round of CNOT gates, and so on in the sequence H-C-P-C-P-C-H-P-C-P-C (where H, C, P stand for Hadamard, CNOT, phase, respectively). One immediate corollary, building on a result by Patel, Markov, and Hayes [17] and improving one by Dehaene and De Moor [18], is that any stabilizer circuit on n qubits has an equivalent circuit with only $O(n^2/\log n)$ gates.

Finally, Sec. VII extends our simulation algorithm to situations beyond the usual one considered in the Gottesman-Knill theorem. For example, we show how to handle mixed states, *without* keeping track of pure states from which the mixed states are obtainable by discarding qubits. We also show how to simulate circuits involving a small number of nonstabilizer gates; or involving arbitrary tensor-product initial states, but only a small number of measurements. Both of these latter two simulations take time that is polynomial in the number of qubits, but exponential in the number of nonstabilizer gates or measurements. Presumably this exponential dependence is necessary, since otherwise we could simulate arbitrary quantum computations in classical subexponential time.

We conclude in Sec. VIII with some directions for further research.

II. PRELIMINARIES

We assume familiarity with quantum computing. This section provides a crash course on the stabilizer formalism, confining attention to those aspects we will need. See Sec. 10.5.1 of Nielsen and Chuang [19] for more details.

Throughout this paper we will use the following four Pauli matrices:

$$I=\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix},\quad X=\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix},$$

$$Y=\begin{pmatrix}0 & -i\\ i & 0\end{pmatrix},\quad Z=\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}.$$

These matrices satisfy the following identities:

$$X^2=Y^2=Z^2=I,$$

$$XY=iZ,\quad YZ=iX,\quad ZX=iY,$$

$$YX=-iZ,\quad ZY=-iX,\quad XZ=-iY.$$

In particular, every two Pauli matrices either commute or anticommute. The rule for whether to include a minus sign is the same as that for quaternions, if we replace (I,X,Y,Z) by $(1,i,j,k)$.

We define the group $\mathcal{P}_n$ of n-qubit *Pauli operators* to consist of all tensor products of n Pauli matrices, together with a multiplicative factor of ± 1 or $\pm i$ (so the total number of operators is $|\mathcal{P}_n|=4^{n+1}$). We omit tensor-product signs for brevity; thus $-YZZI$ should be read $-Y\otimes Z\otimes Z\otimes I$ (we will use $+$ to represent the Pauli group operation). Given two Pauli operators $P=i^k P_1\cdots P_n$ and $Q=i^l Q_1\cdots Q_n$, it is immediate that P commutes with Q if and only if the number of indices $j\in\{1,\ldots,n\}$ such that P_j anticommutes with Q_j is even; otherwise P anticommutes with Q. Also, for all $P\in\mathcal{P}_n$, if P has a phase of ± 1 then $P+P=I\cdots I$, whereas if P has a phase of $\pm i$ then $P+P=-I\cdots I$.

Given a pure quantum state $|\psi\rangle$, we say a unitary matrix U stabilizes $|\psi\rangle$ if $|\psi\rangle$ is an eigenvector of U with eigenvalue 1, or equivalently if $U|\psi\rangle=|\psi\rangle$ where we do not ignore global phase. To illustrate, the following table lists the Pauli matrices and their opposites, together with the unique one-qubit states that they stabilize:

$$X:\ |0\rangle + |1\rangle, \quad -X:\ |0\rangle - |1\rangle,$$

$$Y:\ |0\rangle + i|1\rangle, \quad -Y:\ |0\rangle - i|1\rangle,$$

$$Z:\ |0\rangle, \quad -Z:\ |1\rangle.$$

The identity matrix I stabilizes all states, whereas $-I$ stabilizes no states.

The key idea of the stabilizer formalism is to represent a quantum state $|\psi\rangle$, not by a vector of amplitudes, but by a *stabilizer group*, consisting of unitary matrices that stabilize $|\psi\rangle$. Notice that if U and V both stabilize $|\psi\rangle$ then so do UV and U^{-1}, and thus the set Stab $(|\psi\rangle)$ of stabilizers of $|\psi\rangle$ is a group. Also, it is not hard to show that if $|\psi\rangle \neq |\varphi\rangle$ then $\text{Stab}(|\psi\rangle) \neq \text{Stab}(|\varphi\rangle)$. But why does this strange representation buy us anything? To write down generators for $\text{Stab}(|\psi\rangle)$ (even approximately) still takes exponentially many bits in general by an information-theoretic argument. Indeed stabilizers seem *worse* than amplitude vectors, since they require about 2^{2n} parameters to specify instead of about 2^n.

Remarkably, though, a large and interesting class of quantum states can be specified uniquely by much smaller stabilizer groups—specifically, the intersection of $\text{Stab}(|\psi\rangle)$ with the Pauli group [11–13]. This class of states, which arises in quantum error correction and many other settings, is characterized by the following theorem.

Theorem 1. Given an n-qubit state $|\psi\rangle$, the following are equivalent.

(i) $|\psi\rangle$ can be obtained from $|0\rangle^{\otimes n}$ by CNOT, Hadamard, and phase gates only.

(ii) $|\psi\rangle$ can be obtained from $|0\rangle^{\otimes n}$ by CNOT, Hadamard, phase, and measurement gates only.

(iii) $|\psi\rangle$ is stabilized by exactly 2^n Pauli operators.

(iv) $|\psi\rangle$ is uniquely determined by $S(|\psi\rangle) = \text{Stab}(|\psi\rangle) \cap \mathcal{P}_n$, or the group of Pauli operators that stabilize $|\psi\rangle$.

Because of Theorem 1, we call any circuit consisting entirely of CNOT, Hadamard, phase, and measurement gates a *stabilizer circuit*, and any state obtainable by applying a stabilizer circuit to $|0\rangle^{\otimes n}$ a *stabilizer state*. As a warmup to our later results, the following proposition counts the number of stabilizer states.

Proposition 1. Let N be the number of pure stabilizer states on n qubits. Then

$$N = 2^n \prod_{k=0}^{n-1} (2^{n-k} + 1) = 2^{[1/2+o(1)]n^2}.$$

Proof. We have $N = G/A$, where G is the total number of generating sets and A is the number of equivalent generating sets for a given stabilizer S. To find G, note that there are $4^n - 1$ choices for the first generator M_1 (ignoring overall sign), because it can be anything but the identity. The second generator must commute with M_1 and cannot be I or M_1, so there are $4^n/2 - 2$ choices for M_2. Similarly, M_3 must commute with M_1 and M_2, but cannot be in the group generated by them, so there are $4^n/4 - 4$ choices for it, and so on. Hence, including overall signs,

$$G = 2^n \prod_{k=0}^{n-1} \left(\frac{4^n}{2^k} - 2^k \right) = 2^{n(n+1)/2} \prod_{k=0}^{n-1} (4^{n-k} - 1).$$

Similarly, to find A, note that given S, there are $2^n - 1$ choices for M_1, $2^n - 2$ choices for M_2, $2^n - 4$ choices for M_3, and so on. Thus

$$A = \prod_{k=0}^{n-1} (2^n - 2^k) = 2^{n(n-1)/2} \prod_{k=0}^{n-1} (2^{n-k} - 1).$$

Therefore

$$N = \frac{G}{A} = 2^n \prod_{k=0}^{n-1} \left(\frac{4^{n-k} - 1}{2^{n-k} - 1} \right) = 2^n \prod_{k=0}^{n-1} (2^{n-k} + 1).$$

■

III. EFFICIENT SIMULATION OF STABILIZER CIRCUITS

Theorem 1 immediately suggests a way to simulate stabilizer circuits efficiently on a classical computer. A well-known fact from group theory says that any finite group G has a generating set of size at most $\log_2 |G|$. So if $|\psi\rangle$ is a stabilizer state on n qubits, then the group $S(|\psi\rangle)$ of Pauli operators that stabilize $|\psi\rangle$ has a generating set of size $n = \log_2 2^n$. Each generator takes $2n+1$ bits to specify: 2 bits for each of the n Pauli matrices, and 1 bit for the phase.[1] So the total number of bits needed to specify $|\psi\rangle$ is $n(2n+1)$. What Gottesman and Knill showed, furthermore, is that these bits can be updated in polynomial time after a CNOT, Hadamard, phase, or measurement gate is applied to $|\psi\rangle$. The updates corresponding to unitary gates are very efficient, requiring only $O(n)$ time for each gate.

However, the updates corresponding to measurements are not so efficient. We can decide in $O(n)$ time whether a measurement of qubit a will yield a deterministic or random outcome. If the outcome is random, then updating the state after the measurement takes $O(n^2)$ time, but if the outcome is deterministic, then deciding whether the outcome is $|0\rangle$ or $|1\rangle$ seems to require inverting an $n \times n$ matrix, which takes $O(n^{2.376})$ time in theory [20] but order n^3 time in practice. What that n^3 complexity means is that simulations of, say, 2000-qubit systems would already be prohibitive on a desktop PC, given that measurements are frequent.

This section describes a simulation algorithm by which both deterministic and random measurements can be performed in $O(n^2)$ time. The cost is a factor of 2 increase in the number of bits needed to specify a state. For, in addition to the n stabilizer generators, we now store n "destabilizer" generators, which are Pauli operators that together with the stabilizer generators generate the full Pauli group $\mathcal{P}_n$. So the number of bits needed is $2n(2n+1) \approx 4n^2$.

The algorithm represents a state by a *tableau* consisting of

[1] If $P \in S(|\psi\rangle)$, then P can only have a phase of ± 1, not $\pm i$; for in the latter case $P^2 = -I \cdots I$ would be in $S(|\psi\rangle)$, but we saw that $-I$ does not stabilize anything.

binary variables x_{ij}, z_{ij} for all $i \in \{1,\ldots,2n\}$, $j \in \{1,\ldots,n\}$, and r_i for all $i \in \{1,\ldots,2n\}$:[2]

$$\left(\begin{array}{ccc|ccc|c} x_{11} & \cdots & x_{1n} & z_{11} & \cdots & z_{1n} & r_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{n1} & \cdots & x_{nn} & z_{n1} & \cdots & z_{nn} & r_n \\ \hline x_{(n+1)1} & \cdots & x_{(n+1)n} & z_{(n+1)1} & \cdots & z_{(n+1)n} & r_{n+1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{(2n)1} & \cdots & x_{(2n)n} & z_{(2n)1} & \cdots & z_{(2n)n} & r_{2n} \end{array}\right)$$

Rows 1 to n of the tableau represent the destabilizer generators $R_1,\ldots,R_n$, and rows $n+1$ to $2n$ represent the stabilizer generators $R_{n+1},\ldots,R_{2n}$. If $R_i = \pm P_1 \cdots P_n$, then bits x_{ij}, z_{ij} determine the jth Pauli matrix P_j: 00 means I, 01 means X, 01 means Y, and 10 means Z. Finally, r_i is 1 if R_i has negative phase and 0 if r_i has positive phase. As an example, the 2-qubit state $|00\rangle$ is stabilized by the Pauli operators $+ZI$ and $+IZ$, so a possible tableau for $|00\rangle$ is

$$\left(\begin{array}{cc|cc|c} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right)$$

Indeed, we will take the obvious generalization of the above "identity matrix" to be the standard initial tableau.

The algorithm uses a subroutine called rowsum(h,i), which sets generator h equal to $i+h$. Its purpose is to keep track, in particular, of the phase bit r_h, including all the factors of i that appear when multiplying Pauli matrices. The subroutine is implemented as follows.

rowsum(h,i). Let $g(x_1,z_1,x_2,z_2)$ be a function that takes 4 bits as input, and that returns the exponent to which i is raised (either 0, 1, or -1) when the Pauli matrices represented by $x_1 z_1$ and $x_2 z_2$ are multiplied. More explicitly, if $x_1 = z_1 = 0$ then $g = 0$; if $x_1 = z_1 = 1$ then $g = z_2 - x_2$; if $x_1 = 1$ and $z_1 = 0$ then $g = z_2(2x_2 - 1)$; and if $x_1 = 0$ and $z_1 = 1$ then $g = x_2(1 - 2z_2)$. Then set $r_h := 0$ if

$$2r_h + 2r_i + \sum_{j=1}^{n} g(x_{ij}, z_{ij}, x_{hj}, z_{hj}) \equiv 0 \pmod 4,$$

and set $r_h := 1$ if the sum is congruent to 2 mod4 (it will never be congruent to 1 or 3). Next, for all $j \in \{1,\ldots,n\}$, set $x_{hj} := x_{ij} \oplus x_{hj}$ and set $z_{hj} := z_{ij} \oplus z_{hj}$ (here and throughout, $\oplus$ denotes exclusive-OR).

We now give the algorithm. It will be convenient to add an additional $(2n+1)$st row for "scratch space." The initial state $|0\rangle^{\otimes n}$ has $r_i = 0$ for all $i \in \{1,\ldots,2n+1\}$, and $x_{ij} = \delta_{ij}$ and $z_{ij} = \delta_{(i-n)j}$ for all $i \in \{1,\ldots,2n+1\}$ and $j \in \{1,\ldots,n\}$, where δ_{ij} is 1 if $i = j$ and 0 otherwise. The algorithm proceeds through the gates in order; for each one it does one of the following depending on the gate type.

CNOT gate from control a to target b. For all $i \in \{1,\ldots,2n\}$, set $r_i := r_i \oplus x_{ia} z_{ib}(x_{ib} \oplus z_{ia} \oplus 1)$, $x_{ib} := x_{ib} \oplus x_{ia}$, and $z_{ia} := z_{ia} \oplus z_{ib}$.

Hadamard gate on qubit a. For all $i \in \{1,\ldots,2n\}$, set $r_i := r_i \oplus x_{ia} z_{ia}$ and swap x_{ia} with z_{ia}.

Phase gate on qubit a. For all $i \in \{1,\ldots,2n\}$, set $r_i := r_i \oplus x_{ia} z_{ia}$ and then set $z_{ia} := z_{ia} \oplus x_{ia}$.

Measurement gate of qubit a in standard basis. First check whether there exists a $p \in \{n+1,\ldots,2n\}$ such that $x_{pa} = 1$.

Case I. Such a p exists (if more than one exists, then let p be the smallest). In this case the measurement outcome is random, so the state needs to be updated. This is done as follows. First call rowsum(i,p) for all $i \in \{1,\ldots,2n\}$ such that $i \neq p$ and $x_{ia} = 1$. Second, set entire the $(p-n)$th row equal to the pth row. Third, set the pth row to be identically 0, except that r_p is 0 or 1 with equal probability, and $z_{pa} = 1$. Finally, return r_p as the measurement outcome.

Case II. Such a p does not exist. In this case the outcome is determinate, so measuring the state will not change it; the only task is to determine whether 0 or 1 is observed. This is done as follows. First set the $(2n+1)$st row to be identically 0. Second, call rowsum$(2n+1, i+n)$ for all $i \in \{1,\ldots,n\}$ such that $x_{ia} = 1$. Finally, return r_{2n+1} as the measurement outcome.

Once we interpret the x_{ij}, z_{ij}, and r_i bits for $i \geqslant n+1$ as representing generators of $S(|\psi\rangle)$, and rowsum as representing the group operation in $\mathcal{P}_n$, the correctness of the CNOT, Hadamard, phase, and random measurement procedures follows immediately from previous analyses by Gottesman [13]. It remains only to explain why the determinate measurement procedure is correct. Observe that R_h commutes with R_i if the *symplectic inner product*

$$R_h \cdot R_i = x_{h1} z_{i1} \oplus \cdots \oplus x_{hn} z_{in} \oplus x_{i1} z_{h1} \oplus \cdots \oplus x_{in} z_{hn}$$

equals 0, and anticommutes with R_i if $R_h \cdot R_i = 1$. Using that fact it is not hard to show the following.

Proposition 2. The following are invariants of the tableau algorithm.

(i) $R_{n+1},\ldots,R_{2n}$ generate $S(|\psi\rangle)$, and $R_1,\ldots,R_{2n}$ generate $\mathcal{P}_n$.

(ii) $R_1,\ldots,R_n$ commute.

(iii) For all $h \in \{1,\ldots,n\}$, R_h anticommutes with R_{h+n}.

(iv) For all i, $h \in \{1,\ldots,n\}$ such that $i \neq h$, R_i commutes with R_{h+n}.

Now suppose that a measurement of qubit a yields a determinate outcome. Then the Z_a operator must commute with all elements of the stabilizer, so

$$\sum_{h=1}^{n} c_h R_{h+n} = \pm Z_a$$

for a unique choice of $c_1,\ldots,c_n \in \{0,1\}$. Our goal is to determine the c_h's, since then by summing the appropriate R_{h+n}'s we can learn whether the phase representing the out-

[2] Dehaene and De Moor [18] came up with something like this tableau representation independently, though they did not use it to simulate measurements in $O(n^2)$ time.

come is positive or negative. Notice that for all $i \in \{1, \ldots, n\}$,

$$c_i \equiv \sum_{h=1}^{n} c_h (R_i \cdot R_{h+n}) \equiv R_i \cdot \sum_{h=1}^{n} c_h R_{h+n} \equiv R_i \cdot Z_a \pmod 2$$

by Proposition 2. Therefore by checking whether R_i anticommutes with Z_a—which it does if and only if $x_{ia}=1$—we learn whether $c_i=1$ and thus whether rowsum$(2n+1, i+n)$ needs to be called.

We end this section by explaining how to compute the *inner product* between two stabilizer states $|\psi\rangle$ and $|\varphi\rangle$, given their full tableaus. The inner product is 0 if the stabilizers contain the same Pauli operator with opposite signs. Otherwise it equals $2^{-s/2}$, where s is the minimum, over all sets of generators $\{G_1, \ldots, G_n\}$ for Stab $(|\psi\rangle)$ and $\{H_1, \ldots, H_n\}$ for Stab $(|\varphi\rangle)$, of the number of i for which $G_i \neq H_i$. For example, $\langle XX, ZZ\rangle$ and $\langle ZI, IZ\rangle$ have inner product $1/\sqrt{2}$, since $\langle ZI, IZ\rangle = \langle ZI, ZZ\rangle$. The proof is easy: it suffices to observe that neither the inner product nor s is affected if we transform $|\psi\rangle$ and $|\varphi\rangle$ to $U|\psi\rangle$ and $U|\varphi\rangle$, respectively, for some unitary U such that $U|\psi\rangle = |0\rangle^{\otimes n}$ has the trivial stabilizer. This same observation yields an algorithm to compute the inner product: first transform the tableau of $|\psi\rangle$ to that of $U|\psi\rangle = |0\rangle^{\otimes n}$ using Theorem 3 below; then perform Gaussian elimination on the tableau of $U|\varphi\rangle$ to obtain s. Unfortunately, this algorithm takes order n^3 steps.

IV. IMPLEMENTATION AND EXPERIMENTS

We have implemented the tableau algorithm of Sec. III in a C program called CHP (CNOT-Hadamard-phase), which is available for download.[3] CHP takes as input a program in a simple “quantum assembly language,” consisting of four instructions: `c` a b (apply CNOT gate from control a to target b), `h` a (apply Hadamard gate to a), `p` a (apply phase gate to a), and `m` a (measure a in the standard basis, output the result, and update the state accordingly). Here a and b are nonnegative integers indexing qubits; the maximum a or b that occurs in any instruction is assumed to be $n-1$, where n is the number of qubits. As an example, the following program demonstrates the famous quantum teleportation protocol of Bennett *et al.* [16]:

```
h 1   }
c 1 2 }  EPR pair is prepared (qubit 1 is Alice’s half; qubit 2 is Bob’s half)

c 0 1 }
h 0   }
m 0   }  Alice interacts qubit 0 (the state to be teleported) with her half of the EPR pair
m 1   }

c 0 3 }
c 1 4 }  Alice sends 2 classical bits to Bob

c 4 2 }
h 2   }
c 3 2 }  Bob uses the bits from Alice to recover the teleported state.
h 2   }
```

We also have available CHP programs that demonstrate the Bennett-Wiesner dense quantum coding protocol [15], the GHZ experiment [14], Simon’s algorithm [21], and the Shor 9-qubit quantum error-correcting code [22].

Our main design goal for CHP was high performance with a large number of qubits and frequent measurements. The only reason to use CHP instead of a general-purpose quantum computer simulator such as QUIDD [3] or QCL [2] is performance, so we wanted to leverage that advantage and make thousands of qubits easily simulable rather than just hundreds. Also, the results of Sec. V suggest that classical postprocessing is unavoidable for stabilizer circuits, since stabilizer circuits are not even universal for classical computation. So if we want to simulate (for example) Simon’s algorithm, then one measurement is needed for each bit of the first register. CHP’s execution time will be dominated by these measurements, since as discussed in Sec. III each unitary gate takes only $O(n)$ time to simulate.

Our experimental results, summarized in Fig. 2, show that CHP makes practical the simulation of arbitrary stabilizer circuits on up to about 3000 qubits. Since the number of bits

[3] At www.cs.berkeley.edu/~aaronson/chp.html

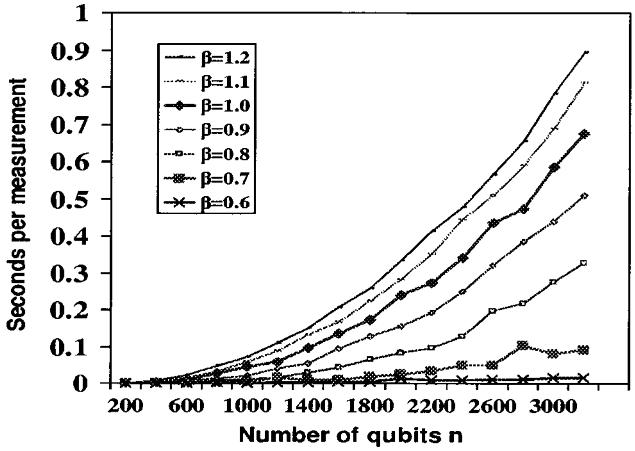

FIG. 2. Average time needed to simulate a measurement after applying $\beta n\log_2 n$ unitary gates to n qubits, on a 650 MHz Pentium III with 256 Mbytes RAM.

needed to represent n qubits grows quadratically in n, the main limitation is available memory. On a machine with 256 Mbytes of random access memory (RAM), CHP can handle up to about 20 000 qubits before virtual memory is needed, in which case thrashing makes its performance intolerable. The original version of CHP required $\sim 8n^2$ bits for memory; we were able to reduce this to $\sim 4n^2$ bits, enabling a 41% increase in the number of qubits for a fixed memory size. More trivially, we obtained an eightfold improvement in memory by storing 8 bits to each byte instead of 1. Not only did that change increase the number of storable qubits by 183%, but it also made CHP about 50% faster—presumably because (1) the rowsum subroutine now needed to exclusive-OR only 1/8 as many bytes, and (2) the memory penalty was reduced. Storing the bits in 32-bit words yielded a further 10% performance gain, presumably because of (1) rather than (2) (since even with byte addressing, a whole memory line is loaded into the cache on a cache miss).

As expected, the experimentally measured execution time per unitary gate grows linearly in n, whereas the time per measurement grows somewhere between linearly and quadratically, depending on the states being measured. Thus the time needed for measurements generally dominates execution time. So the key question is this: what properties of a circuit determine whether the time per measurement is linear, quadratic, or somewhere in between? To investigate this question we performed the following experiment.

We randomly generated stabilizer circuits on n qubits, for n ranging from 200 to 3200 in increments of 200. For each n, we used the following distribution over circuits: Fix a parameter $\beta > 0$; then choose $\lfloor \beta n\log_2 n\rfloor$ random unitary gates: a CNOT gate from control a to target b, a Hadamard gate on qubit a, or a phase gate on qubit a, each with probability 1/3, where a and b are drawn uniformly at random from $\{1,\ldots,n\}$ subject to $a\neq b$. Then measure qubit a for each $a\in\{1,\ldots,n\}$ in sequence.

We simulated the resulting circuits in CHP. For each circuit, we counted the number of seconds needed for all n measurement steps (ignoring the time for unitary gates), then divided by n to obtain the number of seconds per measurement. We repeated the whole procedure for β ranging from 0.6 to 1.2 in increments of 0.1.

There were several reasons for placing measurements at the end of a circuit rather than interspersing them with unitary gates. First, doing so models how many quantum algorithms actually work (apply unitary gates, then measure, then perform classical postprocessing); second, it allowed us to ignore the effect of measurements on subsequent computation; third, it "standardized" the measurement stage, making comparisons between different circuits more meaningful; and fourth, it made simulation harder by increasing the propensity for the measurements to be nontrivially correlated.

The decision to make the number of unitary gates proportional to $n\log n$ was based on the following heuristic argument. The time needed to simulate a measurement is determined by how many times the rowsum procedure is called, which in turn is determined by how many i's there are such that $x_{ia}=1$ (where a is the qubit being measured). Initially $x_{ia}=1$ if and only if $a=i$, so a measurement takes $O(n)$ time. For a random state, by contrast, the expected number of i's such that $x_{ia}=1$ is n by symmetry, so a measurement takes order n^2 time. In general, the more 1's there are in the tableau, the longer measurements take. But where does the transition from linear to quadratic time occur, and how sharp is it?

Consider n people, each of whom initially knows one secret (with no two people knowing the same secret). Each day, two people chosen uniformly at random meet and exchange all the secrets they know. What is the expected number of days until everyone knows everyone else's secrets? Intuitively, the answer is $\Theta(n\log n)$, because any given person has to wait $\Theta(n)$ days between meetings, and at each meeting, the number of secrets he knows approximately doubles (or toward the end, the number of secrets he does not know is approximately halved). Replacing people by qubits and meetings by CNOT gates, one can see why a "phase transition" from a sparse to a dense tableau might occur after $\Theta(n\log n)$ random unitary gates are applied. However, this argument does not pin down the proportionality constant β, so that is what we varied in the experiment.

The results of the experiment are presented in Fig. 2. When $\beta=0.6$, the time per measurement appears to grow roughly linearly in n, whereas when $\beta=1.2$ (meaning that the number of unitary gates has only doubled), the time per measurement appears to grow roughly quadratically, so that running the simulations took 4 h of computing time.[4] Thus, Fig. 2 gives striking evidence for a "phase transition" in simulation time, as increasing the number of unitary gates by only a constant factor shifts us from a regime of simple states that are easy to measure, to a regime of complicated states that are hard to measure. This result demonstrates that CHP's performance depends strongly on the circuit being simulated. Without knowing what sort of tableaus a circuit will produce, all we can say is that the time per measurement will be

[4] Based on our heuristic analysis, we conjecture that for intermediate β, the time per measurement grows as n^c for some $1<c<2$. However, we do not have enough data to confirm or refute this conjecture.

somewhere between linear and quadratic in n.

V. COMPLEXITY OF SIMULATING STABILIZER CIRCUITS

The Gottesman-Knill theorem shows that stabilizer circuits are not universal for quantum computation, unless quantum computers can be simulated efficiently by classical ones. To a computer scientist, this theorem immediately raises a question: where *do* stabilizer circuits sit in the hierarchy of computational complexity theory? In this section we resolve that question, by proving that the problem of simulating stabilizer circuits is complete for a classical complexity class known as $\oplus\mathsf{L}$ (pronounced "parity-L").[5] The usual definition of $\oplus\mathsf{L}$ is as the class of all problems that are solvable by a nondeterministic logarithmic-space Turing machine, that accepts if and only if the total number of accepting paths is odd. But there is an alternate definition that is probably more intuitive to non-computer-scientists. This is that $\oplus\mathsf{L}$ is the class of problems that reduce to simulating a polynomial-size CNOT circuit, i.e., a circuit composed entirely of NOT and CNOT gates, acting on the initial state $|0\cdots0\rangle$. (It is easy to show that the two definitions are equivalent, but this would require us first to explain what the usual definition means.)

From the second definition, it is clear that $\oplus\mathsf{L}\subseteq\mathsf{P}$; in other words, any problem reducible to simulating CNOT circuits is also solvable in polynomial time on a classical computer. But this raises a question: what do we mean by "reducible"? Problem A is reducible to problem B is any instance of problem A can be transformed into an instance of problem B; this means that problem B is "harder" than problem A in the sense that the ability to answer an arbitrary instance of problem B implies the ability to answer an arbitrary instance of problem A (but not necessarily vice versa).

We must, however, insist that the reduction transforming instances of problem A into instances of problem B not be too difficult to perform. Otherwise, we could reduce hard problems to easy ones by doing all the difficult work in the reduction itself. In the case of $\oplus\mathsf{L}$, we *cannot* mean "reducible in polynomial time," which is a common restriction, since then the reduction would be at least as powerful as the problem it reduces to. Instead we require the reduction to be performed in the complexity class L, or logarithmic space—that is, by a Turing machine M that is given a read-only input of size n, and a write-only output tape, but only $O(\log n)$ bits of read/write memory. The reduction works as follows: first M specifies a CNOT circuit on its output tape; then an "oracle" tells M the circuit's output (which we can take to be, say, the value of the first qubit after the circuit is applied); then M specifies another CNOT circuit on its output tape; and so on. A useful result of Hertrampf, Reith, and Vollmer [24] says that this seemingly powerful kind of reduction, in which M can make multiple calls to the CNOT oracle, is actually no more powerful than the kind with only one oracle call. (In complexity language, what [24] showed is that $\oplus\mathsf{L}=\mathsf{L}^{\oplus\mathsf{L}}$: any problem in L with $\oplus\mathsf{L}$ oracle is also in $\oplus\mathsf{L}$ itself.)

It is conjectured that $\mathsf{L}\neq\oplus\mathsf{L}$; in other words, that an oracle for simulating CNOT circuits would let an L machine compute more functions than it could otherwise. Intuitively, this is because writing down the intermediate states of such a circuit requires more than a logarithmic number of read/write bits. Indeed, $\oplus\mathsf{L}$ contains some surprisingly "hard" problems, such as inverting matrices over $\mathbb{GF}_2$ [23]. On the other hand, it is also conjectured that $\oplus\mathsf{L}\neq\mathsf{P}$, meaning that even with an oracle for simulating CNOT circuits, an L machine could not simulate more general circuits with AND and OR gates. As usual in complexity theory, neither conjecture has been proved.

Now define the Gottesman-Knill problem as follows. We are given a stabilizer circuit $\mathcal{C}$ as a sequence of gates of the form CNOT $a\rightarrow b$, Hadamard a, phase a, or measure a, where $a, b\in\{1,\ldots,n\}$ are indices of qubits. The problem is to decide whether qubit 1 will be $|1\rangle$ with certainty after $\mathcal{C}$ is applied to the initial state $|0\rangle^{\otimes n}$. (If not, then qubit 1 will be $|1\rangle$ with probability either $1/2$ or 0.)

Since stabilizer circuits are a generalization of CNOT circuits, it is obvious that the Gottesman-Knill problem is $\oplus\mathsf{L}$-hard (i.e., any $\oplus\mathsf{L}$ problem can be reduced to it). Our result says that the Gottesman-Knill problem is *in* $\oplus\mathsf{L}$. Intuitively, this means that any stabilizer circuit can be simulated efficiently using CNOT gates alone—the additional availability of Hadamard and phase gates gives stabilizer circuits at most a polynomial advantage. In our view, this surprising fact helps to explain the Gottesman-Knill theorem, by providing strong evidence that stabilizer circuits are not even universal for classical computation (assuming, of course, that classical postprocessing is forbidden).

Theorem 2. Gottesman-Knill problem *is in* $\oplus\mathsf{L}$.

Proof. We will show how to solve the Gottesman-Knill problem using a logarithmic-space machine M with an oracle for simulating CNOT circuits. By the result of Hertrampf, Reith, and Vollmer [24] described above, this will suffice to prove the theorem.

By the principle of deferred measurement, we can assume that the stabilizer circuit $\mathcal{C}$ has only a single measurement gate at the end (say of qubit 1), with all other measurements replaced by CNOT gates into ancilla qubits. In the tableau algorithm of Sec. III, let $x_{ij}^{(t)}$, $z_{ij}^{(t)}$, $r_i^{(t)}$ be the values of the variables x_{ij}, z_{ij}, r_i after t gates of $\mathcal{C}$ have been applied. Then M will simulate $\mathcal{C}$ by computing these values. The first task of M is to decide whether the measurement has a determinate outcome—or, equivalently, whether $x_{i1}^{(T)}=0$ for every $i\in\{n+1,\ldots,2n\}$, where T is the number of unitary gates. Observe that in the CNOT, Hadamard, and phase procedures, every update to an x_{ij} or z_{ij} variable replaces it by the sum modulo 2 of one or two other x_{ij} or z_{ij} variables. Also, iterating overall all $t\in\{0,\ldots,T\}$ and $i\in\{1,\ldots,2n\}$ takes only $O(\log n)$ bits of memory. Therefore, despite its memory restriction, M can easily write on its output tape a description of a CNOT circuit that simulates the tableau algorithm using $4n^2$ bits (the r_i's being omitted), and that returns $x_{i1}^{(T)}$ for any desired i. Then to decide whether the measurement outcome is determinate, M simply iterates over all i from $n+1$ to $2n$.

[5] See www.complexityzoo.com for definitions of $\oplus\mathsf{L}$ and several hundred other complexity classes.

The hard part is to decide whether $|0\rangle$ or $|1\rangle$ is measured in case the measurement outcome *is* determinate, for this problem involves the r_i variables, which do not evolve in a linear way as the x_{ij}'s and z_{ij}'s do. Even worse, it involves the complicated-looking and nonlinear rowsum procedure. Fortunately, though, it turns out that the measurement outcome $r_{2n+1}^{(T+1)}$ can be computed by keeping track of a single complex number α. This α is a product of phases of the form ± 1 or $\pm i$, and therefore takes only 2 bits to specify. Furthermore, although the "obvious" ways to compute α use more than $O(\log n)$ bits of memory, M can get around that by making liberal use of the oracle.

First M computes what $r_{2n+1}^{(T+1)}$ *would* be if the CNOT, Hadamard, and phase procedures did not modify the r_i's. Let P be a Pauli matrix with a phase of ± 1 or $\pm i$, which therefore takes 4 bits to specify. Also, let $P_{ij}^{(T)}$ be the Pauli matrix represented by the bits $x_{ij}^{(T)}$, $z_{ij}^{(T)}$ in the usual way: $I=00$, $X=10$, $Y=11$, $Z=01$. Then the procedure is as follows.

```
$\alpha := 1$
for $j := 1$ to $n$
    $P := I$
    for $i := n+1$ to $2n$
        ask oracle for $x_{(i-n)1}^{(T)}$, $x_{ij}^{(T)}$, $z_{ij}^{(T)}$
        if $x_{(i-n)1}^{(T)} = 1$ then $P := P_{ij}^{(T)} P$
    next $i$
    multiply $\alpha$ by the phase of $P$ ($\pm 1$ or $\pm i$)
next $j$
```

The "answer" is 1 if $\alpha=-1$ and 0 if $\alpha=1$ (note that α will never be $\pm i$ at the end). However, M also needs to account for the r_i's, as follows.

```
for $i := n+1$ to $2n$
    ask oracle for $x_{(i-n)1}^{(T)}$
    if $x_{(i-n)1}^{(T)} = 1$
        for $t := 0$ to $T-1$
            if $(t+1)$st gate is a Hadamard or phase gate on $a$
                ask oracle for $x_{ia}^{(t)}$, $z_{ia}^{(t)}$
                if $x_{ia}^{(t)} z_{ia}^{(t)} = 1$ then $\alpha := -\alpha$
            end if
            if $(t+1)$st gate is a CNOT gate from $a$ to $b$
                ask oracle for $x_{ia}^{(t)}$, $z_{ia}^{(t)}$, $x_{ib}^{(t)}$, $z_{ib}^{(t)}$
                if $x_{ia}^{(t)} z_{ib}^{(t)} (x_{ib}^{(t)} \oplus z_{ia}^{(t)} \oplus 1) = 1$ then $\alpha := -\alpha$
            end if
        next $t$
    end if
next $i$
```

The measurement outcome $r_{2n+1}^{(T+1)}$ is then 1 if $\alpha=-1$ and 0 if $\alpha=1$. As described above, the machine M needs only $O(\log n)$ bits to keep track of the loop indices i, j, t, and $O(1)$ additional bits to keep track of other variables. Its correctness follows straightforwardly from the correctness of the tableau algorithm. ■

For a problem to be $\oplus\mathsf{L}$-complete simply means that it is $\oplus\mathsf{L}$-hard *and* in $\oplus\mathsf{L}$. Thus, a corollary of Theorem 2 is that the Gottesman-Knill problem is $\oplus\mathsf{L}$-complete.

VI. CANONICAL FORM

Having studied the simulation of stabilizer circuits, in this section we turn our attention to manipulating those circuits. This task is of direct relevance to quantum computer architecture: because the effects of decoherence build up over time, it is imperative (even more so than for classical circuits) to minimize the number of gates as well as wires and other resources. Even if fault-tolerant techniques will eventually be used to tame decoherence, there remains the bootstrapping problem of building the fault-tolerance hardware. In that regard we should point out that fault-tolerance hardware is likely to consist mainly of CNOT, Hadamard, and phase gates, since the known fault-tolerant constructions (for example, that of Aharonov and Ben-Or [25]) are based on stabilizer codes.

Although there has been some previous work on synthesizing CNOT circuits [17,26,27] and general classical reversible circuits [28,29], to our knowledge there has not been work on synthesizing stabilizer circuits. In this section we prove a *canonical form theorem* that is extremely useful for stabilizer circuit synthesis. The theorem says that given any circuit consisting of CNOT, Hadamard, and phase gates, there exists an equivalent circuit that applies a round of Hadamard gates only, then a round of CNOT gates only, and so on in the sequence H-C-P-C-P-C-H-P-C-P-C. One easy corollary of the theorem is that any tableau satisfying the commutativity conditions of Proposition 2 can be generated by some stabilizer circuit. Another corollary is that any unitary stabilizer circuit has an equivalent circuit with only $O(n^2/\log n)$ gates.

Given two n-qubit unitary stabilizer circuits $\mathcal{C}_1, \mathcal{C}_2$, we say that $\mathcal{C}_1$ and $\mathcal{C}_2$ are *equivalent* if $\mathcal{C}_1(|\psi\rangle)=\mathcal{C}_2(|\psi\rangle)$ for all stabilizer states $|\psi\rangle$, where $\mathcal{C}_i(|\psi\rangle)$ is the final state when $\mathcal{C}_i$ is applied to $|\psi\rangle$.[6] By linearity, it is easy to see that equivalent stabilizer circuits will behave identically on *all* states, not just stabilizer states. Furthermore, there exists a one-to-one correspondence between circuits and tableaus.

Lemma 1. Let $\mathcal{C}_1$, $\mathcal{C}_2$ be unitary stabilizer circuits, and let $\mathcal{T}_1$, $\mathcal{T}_2$ be their respective final tableaus when we run them on the standard initial tableau. Then $\mathcal{C}_1$ and $\mathcal{C}_2$ are equivalent if and only if $\mathcal{T}_1=\mathcal{T}_2$.

Proof. Clearly $\mathcal{T}_1=\mathcal{T}_2$ if $\mathcal{C}_1$ and $\mathcal{C}_2$ are equivalent. For the other direction, it suffices to observe that a unitary stabilizer circuit acts linearly on Pauli operators (that is, rows of the tableau): if it maps P_1 to Q_1 and P_2 to Q_2, then it maps P_1+P_2 to Q_1+Q_2. Since the rows of the standard initial tableau form a basis for $\mathcal{P}_n$, the lemma follows. ■

Our proof of the canonical form theorem will use the following two lemmas.

Lemma 2. Given an n-qubit stabilizer state, it is always possible to apply Hadamard gates to a subset of the qubits so as to make the X matrix have full rank (or, equivalently, make all 2^n basis states have nonzero amplitude).

Proof. We can always perform row additions on the $n \times 2n$ stabilizer matrix without changing the state that it represents. Suppose the X matrix has rank $k<n$; then by Gaussian elimination, we can put the stabilizer matrix in the form

[6] The reason we restrict attention to unitary circuits is simply that, if measurements are included, then it is unclear what it even means for two circuits to be equivalent. For example, does deferring all measurements to the end of a computation preserve equivalence or not?

$$\left(\begin{array}{c|c} A & B \\ 0 & C \end{array}\right)$$

where A is $k\times n$ and has rank k. Then since the rows are linearly independent, C must have rank $n-k$; therefore it has an $(n-k)\times(n-k)$ submatrix C_2 of full rank. Let us permute the columns of the X and Z matrices simultaneously to obtain

$$\left(\begin{array}{cc|cc} A_1 & A_2 & B_1 & B_2 \\ 0 & 0 & C_1 & C_2 \end{array}\right),$$

and then perform Gaussian elimination on the bottom $n-k$ rows to obtain

$$\left(\begin{array}{cc|cc} A_1 & A_2 & B_1 & B_2 \\ 0 & 0 & D & I \end{array}\right).$$

Now commutativity relations imply

$$(A_1 \quad A_2)\begin{pmatrix} D^T \\ I \end{pmatrix} = 0$$

and therefore $A_1D^T=A_2$. Notice that this implies that the $k\times k$ matrix A_1 has full rank, since otherwise the X matrix would have column rank less than k. So applying Hadamard gates to the rightmost $n-k$ qubits yields a state

$$\left(\begin{array}{cc|cc} A_1 & B_2 & B_1 & A_2 \\ 0 & I & D & 0 \end{array}\right)$$

whose X matrix has full rank. ■

Lemma 3. For any matrix $A\in\mathbb{Z}_2^{n\times n}$, there exists a diagonal matrix Λ such that $A+\Lambda$ has full rank.

Proof. Consider using Gaussian elimination to reduce $A+\Lambda$ to an upper-triangular matrix. Initially set $\Lambda:=0$. Then, when the ith row is about to be used to zero out the ith column, if $(A+\Lambda)_{ii}=0$ then set $\Lambda_{ii}:=1$ and continue. Let Λ_{final} be the diagonal matrix that results; then Gaussian elimination reduces $A+\Lambda_{\text{final}}$ to an upper-triangular matrix with all 1's along the diagonal. ■

Say a unitary stabilizer circuit is in canonical form if it consists of 11 rounds in the sequence H-C-P-C-P-C-H-P-C-P-C.

Theorem 3. Any unitary stabilizer circuit has an equivalent circuit in canonical form.

Proof. Divide a $2n\times 2n$ tableau into four $n\times n$ matrices $A=(a_{ij})$, $B=(b_{ij})$, $C=(c_{ij})$, and $D=(d_{ij})$, containing the destabilizer x_{ij} bits, destabilizer z_{ij} bits, stabilizer x_{ij} bits, and stabilizer z_{ij} bits, respectively:

$$\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)$$

(we can ignore the phase bits r_i). Since unitary circuits are reversible, by Lemma 1 it suffices to show how to obtain the standard initial tableau starting from an arbitrary A, B, C, D.[7] We cannot use row additions, since although they leave states invariant they do not in general leave circuits invariant.

The procedure is as follows.

(1) Use Hadamard gates to make C have full rank (this is possible by Lemma 2).

(2) Use CNOT gates to perform Gaussian elimination on C, producing

$$\left(\begin{array}{c|c} A & B \\ \hline I & D \end{array}\right).$$

(3) Use phase gates to make D invertible (this is possible by Lemma 3). Now commutativity of the stabilizer implies that ID^T is symmetric, therefore D is symmetric, therefore $D=MM^T$ for some invertible M.

(4) Use CNOT gates to produce

$$\left(\begin{array}{c|c} A & B \\ \hline M & M \end{array}\right).$$

Note that when we map I to IM, we also map D to $D(M^T)^{-1}=MM^T(M^T)^{-1}=M$.

(5) Apply phase gates to all n qubits to obtain

$$\left(\begin{array}{c|c} A & B \\ \hline M & 0 \end{array}\right).$$

Since M is full rank, there exists some subset S of qubits such that applying two phase gates in succession to every $a\in S$ will preserve the above tableau, but set $r_{n+1}=\cdots=r_{2n}=0$. Apply two phase gates to every $a\in S$.

(6) Use CNOT gates to perform Gaussian elimination on M, producing

$$\left(\begin{array}{c|c} A & B \\ \hline I & 0 \end{array}\right).$$

By commutativity relations, $IB^T=A0^T+I$, therefore $B=I$.

(7) Use Hadamard gates to produce

$$\left(\begin{array}{c|c} I & A \\ \hline 0 & I \end{array}\right).$$

(8) Use phase gates to make A invertible (here we again appeal to Lemma 3). Now commutativity of the destabilizer implies that A is symmetric, therefore $A=NN^T$ for some invertible N.

(9) Use CNOT gates to produce

$$\left(\begin{array}{c|c} N & N \\ \hline 0 & C \end{array}\right).$$

(10) Use phase gates to produce

$$\left(\begin{array}{c|c} N & 0 \\ \hline 0 & C \end{array}\right);$$

then by commutativity relations, $NC^T=I$. Next apply two phase gates each to some subset of qubits in order to preserve the above tableau, but set $r_1=\cdots=r_n=0$.

(11) Use CNOT gates to produce

[7] Actually, this gives the canonical form for the inverse of the circuit, but of course the same argument holds for the inverse circuit too, which is also a stabilizer circuit.

$$\left(\begin{array}{c|c} I & 0 \\ \hline 0 & I \end{array}\right).$$

■

Since Theorem 3 relied only on a tableau satisfying the commutativity conditions, not on its being generated by some stabilizer circuit, an immediate corollary is that any tableau satisfying the conditions is generated by some stabilizer circuit. We can also use Theorem 3 to answer the following question: how many gates are needed for an n-qubit stabilizer circuit in the worst case? Cleve and Gottesman [30] showed that $O(n^2)$ gates suffice for the special case of state preparation, and Gottesman [31] and Dehaene and De Moor [18] showed that $O(n^2)$ gates suffice for stabilizer circuits more generally; even these results were not obvious *a priori*. However, with the help of our canonical form theorem we can show a stronger upper bound.

Corollary 1. Any unitary stabilizer circuit has an equivalent circuit with only $O(n^2/\log n)$ gates.

Proof. Patel, Markov, and Hayes [17] showed that any CNOT circuit has an equivalent CNOT circuit with only $O(n^2/\log n)$ gates. So given a stabilizer circuit $\mathcal{C}$, first put $\mathcal{C}$ into canonical form, then minimize the CNOT segments. Clearly the Hadamard and phase segments require only $O(n)$ gates each. ■

Corollary 1 is easily seen to be optimal by a Shannon counting argument: there are $2^{\Theta(n^2)}$ distinct stabilizer circuits on n qubits, but at most $(n^2)^T$ with T gates.

A final remark: as noted by Moore and Nilsson [27], any CNOT circuit has an equivalent CNOT circuit with $O(n^2)$ gates and parallel depth $O(\log n)$. Thus, using the same idea as in Corollary 1, we obtain that any unitary stabilizer circuit has an equivalent stabilizer circuit with $O(n^2)$ gates and parallel depth $O(\log n)$. (Moore and Nilsson showed this for the special case of stabilizer circuits composed of CNOT and Hadamard gates only.)

VII. BEYOND STABILIZER CIRCUITS

In this section, we discuss generalizations of stabilizer circuits that are still efficiently simulable. The first (easy) generalization, in Sec. VII A, is to allow the computer to be in a mixed rather than a pure state. Mixed states could be simulated by simply purifying the state, and then simulating the purification; but we present an alternative and slightly more efficient strategy.

The second generalization, in Sec. VII B, is to initial states other than the computational basis state. Taken to an extreme, one could even have noncomputable initial states. When combined with arbitrary quantum circuits, such quantum advice is very powerful, although its exact power (relative to classical advice) is unknown [32]. We consider a more modest situation, in which the initial state may include specific ancilla states, consisting of at most b qubits each. The initial state is therefore a tensor product of blocks of b qubits. Given an initial state of this form and general stabilizer circuits, including measurements and classical feedback based on measurement outcomes, universal quantum computation is again possible [33,34]. However, we show that an efficient classical simulation exists, provided only a few measurements are allowed.

The final generalization, in Sec. VII C, is to circuits containing a few nonstabilizer gates. The qualifier "few" is essential here, since it is known that unitary stabilizer circuits plus any additional gate yields a universal set of quantum gates [35,36]. The running time of our simulation procedure is polynomial in n, the number of qubits, but is exponential in the d, the number of nonstabilizer gates.

A. Mixed states

We first present the simulation for mixed states. We allow only *stabilizer mixed states*—that is, states that are uniform distributions over all states in a subspace (or equivalently, all stabilizer states in the subspace) with a given stabilizer of $r < n$ generators. Such mixed states can always be written as the partial trace of a pure stabilizer state, which immediately provides one way of simulating them.

It will be useful to see how to write the density matrix of the mixed state in terms of the stabilizer. The operator $(I+M)/2$, when M is a Pauli operator, is a projection onto the +1 eigenspace of M. Therefore, if the stabilizer of a pure state has generators $M_1,\ldots,M_n$, then the density matrix for that state is

$$\rho = \frac{1}{2^n}\prod_{i=1}^{n}(I+M_i).$$

The density matrix for a stabilizer mixed state with stabilizer generated by $M_1,\ldots,M_r$ is

$$\rho = \frac{1}{2^r}\prod_{i=1}^{r}(I+M_i).$$

To perform our simulation, we find a collection of $2(n-r)$ operators $\overline{X}_i$ and $\overline{Z}_i$ that commute with both the stabilizer and the destabilizer. We can choose them so that $[\overline{X}_i,\overline{X}_j]=[\overline{Z}_i,\overline{Z}_j]=[\overline{X}_i,\overline{Z}_j]=0$ for $i\neq j$, but $\{\overline{X}_i,\overline{Z}_i\}=0$. This can be done by solving a set of linear equations, which in practice takes time $O(n^3)$. If we start with an initial mixed state, we will assume it is of the form $|00\cdots0\rangle\langle00\cdots0|\otimes I$ (so 0 on the first $n-r$ qubits and the completely mixed state on the last r qubits). In that case, we choose $\overline{X}_i = X_{i+r}$ and $\overline{Z}_i = Z_{i+r}$.

We could purify this state by adding $\overline{Z}_i Z_{n+i}$ and $\overline{X}_i X_{n+i}$ to the stabilizer and X_{n+i} and Z_{n+i} to the destabilizer for $i=1,\ldots,r$. Then we could simulate the system by just simulating the evolution of this pure state through the circuit; the extra r qubits are never altered.

A more economical simulation is possible, however, by just keeping track of the original r-generator stabilizer and destabilizer, plus the $2(n-r)$ operators $\overline{X}_i$ and $\overline{Z}_i$. Formally, this allows us to maintain a complete tableau and generalize the $O(n^2)$ tableau algorithm from Sec. III. We place the r generators of the stabilizer as rows $n+1,\ldots,n+r$ of the tableau, and the corresponding elements of the destabilizer as rows $1,\ldots,r$. The new operators $\overline{X}_i$ and $\overline{Z}_i$ $(i=1,\ldots,n-r)$

become rows $r+i$ and $n+r+i$, respectively. Let $\overline{i}=i+n$ if $i \leqslant n$ and $\overline{i}=i-n$ if $i \geqslant n+1$. Then we have that rows R_i and R_j commute unless $i=\overline{j}$, in which case R_i and R_j anticommute.

We can keep track of this new kind of tableau in much the same way as the old kind. Unitary operations transform the new rows the same way as rows of the stabilizer or destabilizer. For example, to perform a CNOT gate from control qubit a to target qubit b, set $x_{ib} := x_{ib} \oplus x_{ia}$ and $z_{ia} := z_{ia} \oplus z_{ib}$, for all $i \in \{1, \ldots, 2n\}$.

Measurement of qubit a is slightly more complex than before. There are now three cases.

Case I. $x_{pa}=1$ for some $p \in \{n+1, \ldots, n+r\}$. In this case Z_a anticommutes with an element of the stabilizer, and the measurement outcome is random. We update as before, for all rows of the tableau.

Case II. $x_{pa}=0$ for all $p > r$. In this case Z_a is in the stabilizer. The measurement outcome is determinate, and we can predict the result as before, by calling rowsum to add up rows r_{n+i} for those i with $x_{ia}=1$.

Case III. $x_{pa}=0$ for all $p \in \{n+1, \ldots, n+r\}$, but $x_{ma}=1$ for some $m \in \{r+1, \ldots, n\}$ or $m \in \{n+r+1, \ldots, 2n\}$. In this case Z_a commutes with all elements of the stabilizer but is not itself in the stabilizer. We get a random measurement result, but a slightly different transformation of the stabilizer than in case I. Observe that row R_m anticommutes with Z_a. This row takes the role of row p from case I, and the row $R_{\overline{m}}$ takes the role of row $p-n$. Update as before with this modification. Then swap rows $n+r+1$ and m and rows $r+1$ and $\overline{m}$. Finally, increase r to $r+1$: the stabilizer has gained a new generator.

Another operation that we might want to apply is discarding the qubit a, which has the effect of performing a partial trace over that qubit in the density matrix. Again, this can be done by simply keeping the qubit in our simulation and not using it in future operations. Here is an alternative: put the stabilizer in a form such that there is at most one generator with an X on qubit a, and at most one with a Z on qubit a. Then drop those two generators (or one, if there is only one total). The remaining generators describe the stabilizer of the reduced mixed state. We also must put the $\overline{X_i}$ and $\overline{Z_i}$ operators in a form where they have no entries in the discarded location, while preserving the structure of the tableau (namely, the commutation relations of Proposition 2). This can also be done in time $O(n^2)$, but we omit the details, as they are rather involved.

B. Nonstabilizer initial states

We now show how to simulate a stabilizer circuit where the initial state is more general, involving nonstabilizer initial states. We allow any number of ancillas in arbitrary states, but the overall ancilla state must be a tensor product of blocks of at most b qubits each. An arbitrary stabilizer circuit is then applied to this state. We allow measurements, but only d of them in total throughout the computation. We do allow classical operations conditioned on the outcomes of measurements, so we also allow polynomial-time classical computation during the circuit.

Let the initial state have density matrix ρ: a tensor product of m blocks of at most b qubits each. Without loss of generality, we first apply the unitary stabilizer circuit U_1, followed by the measurement Z_1 (that is, a measurement of the first qubit in the standard basis). We then apply the stabilizer circuit U_2, followed by measurement Z_2 on the second qubit, and so on up to U_d, Z_d.

We can calculate the probability $p(0)$ of obtaining outcome 0 for the first measurement Z_1 as follows:

$$p(0) = \mathrm{Tr}[(I+Z_1)U_1\rho U_1^{\dagger}]/2 = \mathrm{Tr}[(I+U_1^{\dagger}Z_1U_1)\rho]/2$$
$$= 1/2 + \mathrm{Tr}[(U_1^{\dagger}Z_1U_1)\rho]/2 .$$

But U_1 is a stabilizer operation, so $U_1^{\dagger}Z_1U_1$ is a Pauli matrix, and is therefore a tensor product operation. We also know ρ is a tensor product of blocks of at most b qubits, and the trace of a tensor product is the product of the traces. Let $\rho = \otimes_{j=1}^{m}\rho_j$ and $U_1^{\dagger}Z_1U_1 = \otimes_{j=1}^{m}P_j$ where j ranges over the blocks. Then

$$p(0) = \frac{1}{2} + \prod_{j=1}^{m} \mathrm{Tr}(P_j\rho_j) .$$

Since P_j and ρ_j are both $(2^b \times 2^b)$-dimensional matrices, each $\mathrm{Tr}(P_j\rho_j)$ can be computed in time $O(2^{2b})$.

By flipping an appropriately biased coin, Alice can generate an outcome of the first measurement according to the correct probabilities. Conditioned on this outcome (say of 0), the state of the system is

$$\frac{(I+Z_1)U_1\rho U_1^{\dagger}(1+Z_1)}{4p(0)} .$$

After the next stabilizer circuit U_2, the state is

$$\frac{U_2(I+Z_1)U_1\rho U_1^{\dagger}(1+Z_1)U_2^{\dagger}}{4p(0)} .$$

The probability of obtaining outcome 0 for the second measurement, conditioned on the outcome of the first measurement being 0, is then

$$p(0|0) = \frac{\mathrm{Tr}[(I+Z_2)U_2(I+Z_1)U_1\rho U_1^{\dagger}(1+Z_1)U_2^{\dagger}]}{8p(0)} .$$

By expanding out the eight terms, and then commuting U_1 and U_2 past Z_1 and Z_2, we can write this as

$$\sum_{i=1}^{8} \prod_{j=1}^{m} \mathrm{Tr}(P_{ij}^{(2)}\rho_{ij}) .$$

Each $\mathrm{Tr}(P_{ij}^{(2)}\rho_{ij})$ term can again be computed in time $O(2^{2b})$.

Similarly, the probability of any particular sequence of measurement outcomes $m_1m_2\cdots m_d$ can be written as a sum

$$p(m_1m_2\cdots m_d) = \sum_{i=1}^{2^{2d-1}} \prod_{j=1}^{m} \mathrm{Tr}(P_{ij}^{(d)}\rho_{ij}) ,$$

where each trace can be computed in time $O(2^{2b})$. It follows that the probabilities of the two outcomes of the dth measurement can be computed in time $O(m2^{2b+2d})$.

C. Nonstabilizer gates

The last case that we consider is that of a circuit containing d nonstabilizer gates, each of which acts on at most b qubits. We allow an unlimited number of Pauli measurements and unitary stabilizer gates, but the initial state is required to be a stabilizer state—for concreteness, $|0\rangle^{\otimes n}$.

To analyze this case, we examine the density matrix ρ_t at the tth step of the computation. Initially, ρ_0 is a stabilizer state whose stabilizer is generated by some $M_1, \ldots, M_n$, so we can write it as

$$\rho = \frac{1}{2^n}(I + M_1)(I + M_2) \cdots (I + M_n).$$

If we perform a stabilizer operation, the M_i's become a different set of Pauli operators, but keeping track of them requires at most $n(2n+1)$ bits at any given time [or $2n(2n+1)$ if we include the destabilizer]. If we perform a measurement, the M_i's change in a more complicated way, but remain Pauli group elements.

Now consider a single nonstabilizer gate U. Expanding U in terms of Pauli operations P_i,

$$U\rho U^\dagger = \frac{1}{2^n}\left(\sum_i c_i P_i\right)\prod_j (I + M_j)\left(\sum_k c_k^* P_k\right)$$
$$= \frac{1}{2^n}\sum_{i,k} c_i c_k^* P_i P_k \prod_j (I + (-1)^{M_j \cdot P_k} M_j).$$

Here $M_j \cdot P_k$ is the symplectic inner product between the corresponding vectors, which is 0 whenever M_j and P_k commute and 1 when they anticommute. In what follows, let $c_{ik} = c_i c_k^*$ and $P_{ik} = P_i P_k$. Then we can write the density matrix after U as a sum of terms, each described by a Pauli matrix P_{ik} and a vector of eigenvalues for the stabilizer. Since U and $U^\dagger$ each act on at most b qubits, there are at most 4^{2b} terms in this sum.

If we apply a stabilizer gate to this state, all of the Pauli matrices in the decomposition are transformed to other Pauli matrices, according to the usual rules. If we perform another nonstabilizer gate, we can again expand it in terms of Pauli matrices, and put it in the same form. The new gate can act on b new qubits, however, giving us more terms in the sum. After d such operations, we thus need to keep track of at most 4^{2bd} complex numbers (the coefficients c_{ik}), 4^{bd} strings each of $2n$ bits (the Pauli matrices P_{ik}), and 4^{bd} strings each of n bits (the inner products $M_j \cdot P_k$). We also need to keep track of the stabilizer generators $M_1, \ldots, M_n$, and it will be helpful to also keep track of the destabilizer, for a total of an additional $2n(2n+1)$ bits.

The above allows us to describe the evolution when there are no measurements. What happens when we perform a measurement? Consider the unnormalized density matrix corresponding to outcome 0 for measurement of the Pauli operator Q:

$$\rho(0) = \frac{1}{2^{n+2}} Q^+ \sum_{i,k} c_{ik} P_{ik} \prod_j [I + (-1)^{M_j \cdot P_k} M_j] Q^+,$$

where here and throughout we let $Q^+ = I + Q$ and $Q^- = I - Q$. As usual, either Q commutes with everything in the stabilizer, or Q anticommutes with some element of the stabilizer. (However, the measurement outcome can be indeterminate in both cases, and may have a nonuniform distribution.) In the first case, we can rewrite the density matrix as

$$\rho(0) = \frac{1}{2^{n+2}} \sum_{i,k} c_{ik} Q^+ P_{ik} Q^+ \prod_j [I + (-1)^{M_j \cdot P_k} M_j].$$

But $Q^+ P_{ik} Q^+ = 2 P_{ik} Q^+$ if P_{ik} and Q commute, and $Q^+ P_{ik} Q^+ = Q^+ Q^- P_{ik} = 0$ if P_{ik} and Q anticommute. Furthermore, as usual, as Q commutes with everything in the stabilizer, Q is actually in the stabilizer, so projecting on Q^+ either is redundant (if Q has eigenvalue +1) or annihilates the state (if Q has eigenvalue −1). Therefore, we can see that $\rho(0)$ has the same form as before:

$$\rho(0) = \frac{1}{2^n} \sum_{i,k} c_{ik} P_{ik} \prod_j [I + (-1)^{M_j \cdot P_k} M_j],$$

where now the sum over i is only over those P_{ik}'s that commute with Q, and the sum over k is only over those P_k's that give eigenvalue +1 for Q.

When Q anticommutes with an element of the stabilizer, we can change our choice of generators so that Q commutes with all of the generators except for M_1. Then we write $\rho(0)$ as

$$\rho(0) = \frac{1}{2^{n+2}} \sum_{i,k} c_{ik} Q^+ P_{ik} [I + (-1)^{M_j \cdot P_k} M_1] Q^+ \Lambda_k$$
$$= \frac{1}{2^{n+2}} \sum_{i,k} c_{ik} Q^+ P_{ik} [Q^+ + (-1)^{M_j \cdot P_k} Q^- M_1] \Lambda_k,$$

where

$$\Lambda_k = \prod_{j>1} [I + (-1)^{M_j \cdot P_k} M_j].$$

If P_{ik} and Q commute, then we keep only the first term Q^+ in the square brackets in the second line of the equation for $\rho(0)$. If P_{ik} and Q anticommute, we keep only the second term $Q^- M_1$ in the square brackets. In either case, we can rewrite the density matrix in the same kind of decomposition:

$$\rho(0) = \frac{1}{2^n} \sum_{i,k} c_{ik} P_{ik} Q^+ \prod_{j>1} [I + (-1)^{M_j \cdot P_k} M_j],$$

where Q has replaced M_1 in the stabilizer, and any P_{ik} that anticommutes with Q has been replaced by $P_{ik} M_1$, its corresponding c_{ik} replaced by $(-1)^{M_j \cdot P_k} c_{ik}$.

Therefore, we can always write the density matrix after the measurement in the same kind of sum decomposition as before, with no more terms than there were before the measurement. The density matrices are unnormalized, so we need to calculate $\mathrm{Tr}\rho(0)$ to determine the probability of obtaining outcome 0. Computing the trace of a single term is straightforward: it is 0 if P_{ik} is not in the stabilizer and $\pm 2^n c_{ik}$ if P_{ik} is in the stabilizer (with + or − determined by the eigenvalue of P_{ik}). To calculate $\mathrm{Tr}\rho(0)$, we just need to sum the traces of the 4^{2bd} individual terms. We then choose a random number to determine the actual outcome. Thereafter, we only need to keep track of $\rho(0)$ or $\rho(1)$, which we can easily renormalize to have unit trace. Overall, this simulation therefore takes time and space $O(4^{2bd}n+n^2)$.

VIII. OPEN PROBLEMS

(1) Iwama, Kambayashi, and Yamashita [26] gave a set of *local transformation rules* by which any CNOT circuit (that is, a circuit consisting solely of CNOT gates) can be transformed into any equivalent CNOT circuit. For example, a CNOT gate from a to b followed by another CNOT gate from a to b can be replaced by the identity, and a CNOT gate from a to b followed by a CNOT gate from c to d can be replaced by a CNOT gate from c to d followed by a CNOT gate from a to b, provided that $a \neq d$ and $b \neq c$. Using Theorem 3, can we similarly give a set of local transformation rules by which any unitary stabilizer circuit can be transformed into any equivalent unitary stabilizer circuit? Such a rule set could form the basis of an efficient heuristic algorithm for minimizing stabilizer circuits.

(2) Can the tableau algorithm be modified to compute measurement outcomes in only $O(n)$ time? (In case the measurement yields a random outcome, updating the state might still take order n^2 time.)

(3) In Theorem 3, is the 11-round sequence H-C-P-C-P-C-H-P-C-P-C really necessary, or is there a canonical form that uses fewer rounds? Note that if we are concerned only with state preparation, and not with how a circuit behaves on any initial state other than the standard one, then the five-round sequence H-P-C-P-H is sufficient.

(4) Is there a set of quantum gates that is neither universal for quantum computation, nor classically simulable in polynomial time? Shi [37] has shown that if we generalize stabilizer circuits by adding *any* 1- or 2-qubit gate not generated by CNOT, Hadamard, and phase gates, then we immediately obtain a universal set.

(5) What is the computational power of stabilizer circuits with arbitrary tensor-product initial states, but measurements delayed until the end of the computation? It is known that, if we allow classical postprocessing and control of future quantum operations conditioned on measurement results, then universal quantum computation is possible [33,34]. However, if all measurements are delayed until the end of the computation, then the quantum part of such a circuit (though not the classical postprocessing) can be compressed to constant depth. On the other hand, Terhal and DiVincenzo [38] have given evidence that even constant-depth quantum circuits might be difficult to simulate classically.

(6) Is there an efficient algorithm that, given a CNOT or stabilizer circuit, produces an equivalent circuit of (approximately) minimum size? Would the existence of such an algorithm have unlikely complexity consequences? This might be related to the hard problem of proving superlinear lower bounds on CNOT or stabilizer circuit size for explicit functions.

ACKNOWLEDGMENTS

We thank John Kubiatowicz, Michael Nielsen, Isaac Chuang, Cris Moore, and George Viamontes for helpful discussions, and Andrew Cross for fixing an error in the manuscript and software. S.A. was supported by NSF and DARPA. D.G. is supported by funds from NSERC of Canada, and by the CIAR in the Quantum Information Processing program.

[1] *Quantum Monte Carlo Methods in Equilibrium and Nonequilibrium Systems*, edited by M. Suzuki (Springer, Berlin, 1986).
[2] B. Oemer, http://tph.tuwien.ac.at/~oemer/qcl.html
[3] G. F. Viamontes, I. L. Markov, and J. P. Hayes, Quantum Inf. Process. **2**, 347 (2004).
[4] G. F. Viamontes, M. Rajagopalan, I. L. Markov, and J. P. Hayes, in *Proceedings of the Asia and South-Pacific Design Automation Conference* (IEEE Computer Society, Los Alamitos, CA, 2003), p. 295.
[5] K. M. Obenland and A. M. Despain, *High Performance Computing* (IEEE Computer Society, Los Alamitos, CA, 1998).
[6] G. Vidal, Phys. Rev. Lett. **91**, 147902 (2003).
[7] G. Vidal, e-print quant-ph/0310089.
[8] L. G. Valiant, in *Proceedings of the ACM Symposium on Theory of Computing* (ACM, New York, 2001), p. 114.
[9] B. M. Terhal and D. P. DiVincenzo, Phys. Rev. A **65**, 032325 (2002).
[10] C. H. Bennett, D. P. DiVincenzo, J. A. Smolin, and W. K. Wootters, Phys. Rev. A **54**, 3824 (1996).
[11] A. R. Calderbank, E. M. Rains, P. W. Shor, and N. J. A. Sloane, Phys. Rev. Lett. **78**, 405 (1997).
[12] D. Gottesman, Phys. Rev. A **54**, 1862 (1996).
[13] D. Gottesman, e-print quant-ph/9807006.
[14] D. M. Greenberger, M. A. Horne, and A. Zeilinger, *Bell's Theorem, Quantum Theory, and Conceptions of the Universe* (Kluwer, Dordrecht, 1989), p. 73.
[15] C. H. Bennett and S. J. Wiesner, Phys. Rev. Lett. **69**, 2881 (1992).
[16] C. H. Bennett, G. Brassard, C. Crepeau, R. Jozsa, A. Peres, and W. Wootters, Phys. Rev. Lett. **70**, 1895 (1993).
[17] K. N. Patel, I. L. Markov, and J. P. Hayes, e-print quant-ph/0302002.
[18] J. Dehaene and B. De Moor, Phys. Rev. A **68**, 042318 (2003).
[19] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, 2000).
[20] D. Coppersmith and S. Winograd, J. Symb. Comput. **9**, 251 (1990).
[21] D. R. Simon, SIAM J. Comput. **26**, 1474 (1997).
[22] P. W. Shor, Phys. Rev. A **52**, R2493 (1995).

[23] C. Damm, Inf. Process. Lett. **36**, 247 (1990).

[24] U. Hertrampf, S. Reith, and H. Vollmer, Inf. Process. Lett. **75**, 91 (2000).

[25] D. Aharonov and M. Ben-Or, in *Proceedings of the ACM Symposium on Theory of Computing* (ACM, New York, 1997), p. 176.

[26] K. Iwama, Y. Kambayashi, and S. Yamashita, in Proceedings of Design Automation Conference (2002), p. 419.

[27] C. Moore and M. Nilsson, SIAM J. Comput. **31**, 799 (2002).

[28] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, IEEE Trans. Comput.-Aided Des. **22**, 710 (2003).

[29] J.-S. Lee, Y. Chung, J. Kim, and S. Lee, e-print quant-ph/9911053.

[30] R. Cleve and D. Gottesman, Phys. Rev. A **56**, 76 (1997).

[31] D. Gottesman, Phys. Rev. A **57**, 127 (1998).

[32] S. Aaronson, in *Proceedings of the IEEE Conference on Computational Complexity* (IEEE Computer Society, Los Alamitos, CA, 2004), p. 320.

[33] P. W. Shor, in *Proceedings of the IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society, Los Alamitos, CA, 1996), p. 56.

[34] D. Gottesman and I. Chuang, Nature (London) **402**, 390 (1999).

[35] G. Nebe, E. M. Rains, and N. J. A. Sloane, Designs, Codes, Cryptogr. **24**, 99 (2001).

[36] R. Solovay (unpublished).

[37] Y. Shi, Quantum Inf. Comput. **3**, 84 (2003).

[38] B. M. Terhal and D. P. DiVincenzo, Quantum Inf. Comput. **4**, 134 (2004).

[39] Isaac Chung (private communication).