Quipu: High-performance Simulation of Quantum Circuits using Stabilizer Frames

Héctor J. García Igor L. Markov
University of Michigan, EECS, Ann Arbor, MI 48109-2121
{hjgarcia, imarkov}@eecs.umich.edu

***Abstract*—As quantum information processing gains traction, its simulation becomes increasingly significant for engineering purposes – evaluation, testing and optimization – as well as for theoretical research. Generic quantum-circuit simulation appears intractable for conventional computers. However, Gottesman and Knill identified an important subclass, called *stabilizer circuits*, which can be simulated efficiently using group-theory techniques. Practical circuits enriched with quantum error-correcting codes and fault-tolerant procedures are dominated by stabilizer subcircuits and contain a relatively small number of non-stabilizer components. Therefore, we develop new group-theory data structures and algorithms to simulate such circuits. *Stabilizer frames* offer more compact storage than previous approaches but requires more sophisticated bookkeeping. Our implementation, called `Quipu`, simulates certain quantum arithmetic circuits (e.g., ripple-carry adders) in polynomial time and space for equal superpositions of n-qubits. On such instances, known linear-algebraic simulation techniques, such as the (state-of-the-art) BDD-based simulator `QuIDDPro`, take exponential time. We simulate various quantum Fourier transform and quantum fault-tolerant circuits with `Quipu`, and the results demonstrate that our stabilizer-based technique outperforms `QuIDDPro` in all cases.**

I. Introduction

Quantum information processing manipulates quantum states rather than conventional 0-1 bits. It has been demonstrated with a variety of physical technologies (NMR, ion traps, Josephson junctions in superconductors, linear and non-linear optics) and used in recently developed commercial products. Furthermore, it offers a unique opportunity for EDA research to assist in scientific research. Shor's factoring algorithm [17] and Grover's search algorithm [8] apply the principles of quantum information to carry out computation *asymptotically* more efficiently than conventional computers. These developments fueled research efforts to design, build and program scalable quantum computers. Due to the high volatility of quantum information, quantum error-correcting codes (QECC) and effective fault-tolerant (FT) architectures are necessary to build reliable quantum computers. Most quantum algorithms are described in terms of *quantum circuits* and, just like conventional digital circuits, require functional simulation to determine the best FT design choices given limited resources. Simulating quantum circuits on a conventional computer is a difficult problem. The matrices representing quantum gates, and the vectors that model quantum states grow exponentially with an increase in the number of *qubits* – the quantum analogue of the classical bit. Several software packages have been developed for quantum-circuit simulation including Oemer's Quantum Computation Language (`QCL`) [13] and Viamontes' Quantum Information Decision Diagrams (QuIDD) implemented in the `QuIDDPro` package [18]. While `QCL` simulates circuits directly using state vectors, `QuIDDPro` uses a variant of binary decision diagrams to store state vectors more compactly in some cases. Since the state-vector representation requires excessive computational resources in general, simulation-based reliability studies (e.g. simulated fault-injection analysis) of quantum FT architectures using general-purpose simulators has been limited to small quantum circuits [3]. Therefore, *designing fast simulation techniques that target quantum FT circuits facilitates more robust reliability analysis of larger quantum circuits*.

This work was sponsored in part by the Air Force Research Laboratory under agreement FA8750-11-2-0043.

Stabilizer circuits and states. Gottesman [7] and Knill identified an important subclass of quantum circuits, called *stabilizer circuits*, which can be simulated efficiently on classical computers. Stabilizer circuits are exclusively composed of *stabilizer gates* – controlled-NOT, Hadamard and Phase gates (Figure 1) followed by one-qubit measurements in the computational basis. Such circuits are applied to a computational basis state (usually $|00...0\rangle$) and produce output states known as *stabilizer states*. Because of their extensive applications in QECC and FT architectures, stabilizer circuits have been studied heavily [1], [7]. Stabilizer circuits can be simulated in *polynomial-time* by keeping track of the Pauli operators that stabilize[1] the quantum state. Such *stabilizer operators* are maintained during simulation and uniquely represent stabilizer states up to an unobservable global phase.[2] Therefore, this technique offers an *exponential improvement* over the computational resources needed to simulate stabilize circuits using vector-based representations.

Aaronson and Gottesman [1] proposed an improved technique that uses a bit-vector representation to simulate stabilizer circuits. Aaronson implemented this simulation technique in his `CHP` software package. Compared to other vector-based simulators (`QuIDDPro`, `QCL`) the technique in [1] does not maintain the global phase of a state and simulates each stabilizer gate in $\Theta(n)$ time using $\Theta(n^2)$ space. The overall runtime of `CHP` is dominated by the number of measurement gates, which require $O(n^2)$ time to simulate.

Stabilizer-based simulation of generic circuits. We propose a generalization of the stabilizer formalism that admits simulation of *non-stabilizer gates* such as Toffoli[3] gates. This line of research was first outlined in [1], where the authors describe a stabilizer-based representation that stores an arbitrary quantum state as a sum of density-matrix[4] terms. In contrast, we *store arbitrary states as superpositions*[5] *of stabilizer states*. Such superpositions are stored more compactly than the approach from [1], although we do not handle density matrices. Another key difference is that *our approach explicitly maintains the global phase of each stabilizer state* because in a superposition such phases become relative. We store stabilizer-state superpositions compactly using our proposed *stabilizer frame* data structure. To speed up relevant algorithms, we *store generator sets for each stabilizer frame in row-echelon form* to avoid expensive Gaussian elimination during simulation. The main advantages of using stabilizer-state superpositions to simulate quantum circuits are:

[1] An operator U is said to stabilize a state iff $U|\psi\rangle = |\psi\rangle$.

[2] According to quantum physics, the global phase $exp(i\theta)$ of a quantum state is unobservable and does not need to be simulated.

[3] The Toffoli gate is a 3-bit gate that maps (a, b, c) to $(a, b, c \oplus (ab))$.

[4] Density matrices are self-adjoint positive-semidefinite matrices of trace 1.0, that describe the statistical state of a quantum system [11].

[5] A superposition is a norm-1 linear combination of terms.

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \quad P = \begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix} \quad CNOT = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}$$

Fig. 1. Stabilizer gates: Hadamard (H), Phase (P), controlled-NOT (CNOT).

(i) *Stabilizer subcircuits are simulated with high efficiency.*
(ii) *Superpositions can be restructured and compressed on the fly during simulation to reduce resource requirements.*

Our stabilizer-based technique simulates certain quantum arithmetic circuits in polynomial time and space for input states consisting of unbiased superpositions of computational-basis states. On such instances, known generic simulation techniques take exponential time. We simulate various quantum Fourier transform and quantum FT circuits, and the results demonstrate that our data structure leads to orders-of-magnitude improvement in runtime and memory as compared to state-of-the-art simulators.

In the remaining part of this document, we assume a superficial familiarity with quantum computing, as outlined in [11] and EDA publications such as [16]. Section II describes key concepts related to quantum-circuit simulation and the stabilizer formalism. In Section III, we introduce stabilizer frames and describe in detail our simulation flow implemented in `Quipu`. In Section IV, we discuss our empirical validation of `Quipu` and comparisons with state-of-the-art simulators. Section V closes with concluding remarks.

II. Background and Previous Work

Quantum information processes, including quantum algorithms, are often modeled using *quantum circuits* and are represented by diagrams, just like conventional digital circuits [11], [18]. Quantum circuits are sequences of *gate operations* that act on some register of *qubits* – the basic unit of information in a quantum system. A single qubit is described by a quantum state $|\psi\rangle$, which is a two-dimensional vector over the complex numbers. In contrast to classical bits, qubits can be in a superposition of both the 0 and 1 states. Formally, $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$, where $|0\rangle = (1,0)^\top$ and $|1\rangle = (0,1)^\top$ are the two-dimensional *computational basis states* and α_i are *probability amplitudes* that satisfy $|\alpha_0|^2 + |\alpha_1|^2 = 1$. An n-qubit register is the tensor product of n single qubits and thus is modeled by a complex vector $|\psi^n\rangle = |\psi_1\rangle \otimes \cdots \otimes |\psi_n\rangle = \sum_{i=0}^{2^n-1} \alpha_i |b_i\rangle$, where each b_i is a binary string representing the value i of each basis state. Furthermore, $|\psi^n\rangle$ satisfies $\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$. Each gate operation or *quantum gate* is a *unitary matrix* that operates on a small subset of the qubits in a register. For example, the quantum analogue of a NOT gate is the operator $X = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$,

$$\alpha_0 |00\rangle + \alpha_1 |10\rangle \xrightarrow{X \otimes I} \alpha_0 |10\rangle + \alpha_1 |00\rangle$$

Similarly, the two-qubit CNOT operator flips the second qubit (target) iff the first qubit (control) is set to 1, e.g.,

$$\alpha_0 |00\rangle + \alpha_1 |10\rangle \xrightarrow{CNOT} \alpha_0 |00\rangle + \alpha_1 |11\rangle$$

Another operator of particular importance is the Hadamard (H) gate. This gate is frequently used to put a qubit in a superposition of computational-basis states, e.g.,

$$\alpha_0 |00\rangle + \alpha_1 |10\rangle \xrightarrow{I \otimes H} (\alpha_0 |00\rangle + \alpha_0 |01\rangle + \alpha_1 |10\rangle + \alpha_1 |11\rangle)/\sqrt{2}$$

Note that the H gate generates *unbiased* superpositions in the sense that the squares of the absolute value of the amplitudes are equal. The dynamics involved in observing or measuring a quantum state are described by non-unitary *projection operators*. There are different types of quantum measurements, but the one most pertinent to our discussion are *measurements in the computational basis*, i.e., measurements with respect to the $|0\rangle$ or $|1\rangle$ basis states. The projection operators for such measurements are $P_0 = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$ and $P_1 = \left(\begin{smallmatrix} 0 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, respectively. The probability $p(x)$ of obtaining outcome $x \in \{0,1\}$ on the j^{th} qubit of state $|\psi\rangle$ is given by the inner product $\langle\psi| P_x^j |\psi\rangle$, where $\langle\psi|$ is the conjugate transpose of $|\psi\rangle$. For example, suppose we want to measure $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$ in the $|1\rangle$ basis:

$$p(1) = (\alpha_0^*, \alpha_1^*) P_1 (\alpha_0, \alpha_1)^\top = (0, \alpha_1^*)(\alpha_0, \alpha_1)^\top = |\alpha_1|^2$$

Cofactors of quantum states. The output states obtained after performing computational-basis measurements are called *cofactors*, and are orthogonal states of the form $|0\rangle |\psi_0\rangle$ and $|1\rangle |\psi_1\rangle$. We denote the $|0\rangle$- and $|1\rangle$-cofactor by $|\psi_{j=0}\rangle$ and $|\psi_{j=1}\rangle$, respectively, where j is the index of the measured qubit. One can also consider *iterated cofactors*, such as *double cofactors* $|\psi_{qr=00}\rangle$, $|\psi_{qr=01}\rangle$, $|\psi_{qr=10}\rangle$ and $|\psi_{qr=11}\rangle$. Cofactoring with respect to all qubits produces amplitudes of individual basis vectors.

A. Quantum circuits and simulation

To simulate a quantum circuit $\mathcal{C}$, we first initialize the quantum system to some desired state $|\psi\rangle$ (usually a basis state). $|\psi\rangle$ can be represented using a fixed-size data structure (e.g., an array of 2^n complex numbers) or a variable-size data structure (e.g., algebraic decision diagram). We then track the evolution of $|\psi\rangle$ via its internal representation as the gates in $\mathcal{C}$ are applied until one obtains the output state $\mathcal{C} |\psi\rangle$ [1], [11], [18]. Most quantum-circuit simulators [5], [12], [13], [18] support some form of the linear-algebraic operations described earlier. The drawback of such simulators is that their runtime grows exponentially in the number of qubits. This holds true not only in the worst case but also in many practical applications involving arithmetic and FT circuits.

Gottesman developed a simulation method involving the *Heisenberg model* [7] often used by physicists to describe atomic phenomena. *In this model, one keeps track of the symmetries of an object rather than represent the object explicitly*. In the context of quantum-circuit simulation, this model represents quantum states by their symmetries, rather than complex vectors. The symmetries are operators for which these states are 1-eigenvectors. Algebraically, symmetries form *group* structures, which can be specified compactly by group generators.

B. The stabilizer formalism

A unitary operator U *stabilizes* a state $|\psi\rangle$ iff $|\psi\rangle$ is a 1–eigenvector of U, i.e., $U |\psi\rangle = |\psi\rangle$. We are interested in operators U derived from the Pauli matrices: $X = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right), Y = \left(\begin{smallmatrix} 0 & -i \\ i & 0 \end{smallmatrix}\right), Z = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$, and the identity $I = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$. The one-qubit states stabilized by the Pauli matrices are:

$$\begin{array}{llll} X: & (|0\rangle + |1\rangle)/\sqrt{2} & -X: & (|0\rangle - |1\rangle)/\sqrt{2} \\ Y: & (|0\rangle + i|1\rangle)/\sqrt{2} & -Y: & (|0\rangle - i|1\rangle)/\sqrt{2} \\ Z: & |0\rangle & -Z: & |1\rangle \end{array}$$

Observe that I stabilizes all states and $-I$ does not stabilize any state. Thus, the entangled state $(|00\rangle + |11\rangle)/\sqrt{2}$ is stabilized by the Pauli operators $X \otimes X$, $-Y \otimes Y$, $Z \otimes Z$ and $I \otimes I$. As shown in Table I, it turns out that the Pauli matrices along with I and the multiplicative factors ± 1, $\pm i$, form a *closed group* under matrix multiplication [11]. Formally, the *Pauli group* $\mathcal{G}_n$ on n qubits consists of the n-fold tensor product of Pauli matrices, $P = i^k P_1 \otimes \cdots \otimes P_n$ such that $P_j \in \{I, X, Y, Z\}$ and $k \in \{0, 1, 2, 3\}$. For brevity, the tensor-product symbol is often omitted so that P is denoted by a string of I, X, Y and Z characters or *Pauli literals* and a separate integer value k for the phase i^k. This string-integer pair representation allows us to compute the product of Pauli operators without explicitly computing the tensor products,[6] e.g., $(-IIXI)(iIYII) = -iIYXI$. Since $|\mathcal{G}_n| = 4^{n+1}$, $\mathcal{G}_n$ can have at

[6]This holds true due to the identity: $(A \otimes B)(C \otimes D) = (AC \otimes BD)$.

TABLE I
Multiplication table for Pauli matrices. Shaded cells indicate anticommuting products.

	I	X	Y	Z
I	I	X	Y	Z
X	X	I	iZ	$-iY$
Y	Y	$-iZ$	I	iX
Z	Z	iY	$-iX$	I

most $\log_2 \mid \mathcal{G}_n \mid = \log_2 4^{n+1} = 2(n+1)$ irredundant generators [11]. The key idea behind the stabilizer formalism is to represent an n-qubit quantum state $|\psi\rangle$ by its *stabilizer group* $S(|\psi\rangle)$ – the subgroup of $\mathcal{G}_n$ that stabilizes $|\psi\rangle$. One can show that, if $|S(|\psi\rangle)| = 2^n$, the group uniquely specifies $|\psi\rangle$. In this case, $|\psi\rangle$ belongs to an important class of quantum states called *stabilizer states*. Furthermore, $S(|\psi\rangle)$ itself is specified by only $\log_2 2^n = n$ *irredundant stabilizer generators*. Therefore, an arbitrary n-qubit stabilizer state can be represented by a *stabilizer matrix* $\mathcal{M}$ whose rows represent a set of generators $Q_1, \ldots, Q_n$ for $S(|\psi\rangle)$. (Hence we use the terms *generator set* and *stabilizer matrix* interchangeably.) Since each Q_i is a string of n Pauli literals, the size of the matrix is $n \times n$. The phases of each Q_i are stored separately using a vector of n integers. For example, one can show that $|\psi\rangle = (|00\rangle + |11\rangle)/\sqrt{2}$ is uniquely specified by any of the following matrices: $\mathcal{M}_1 = \begin{smallmatrix}+\\+\end{smallmatrix}\left[\begin{smallmatrix}XX\\ZZ\end{smallmatrix}\right]$, $\mathcal{M}_2 = \begin{smallmatrix}+\\-\end{smallmatrix}\left[\begin{smallmatrix}XX\\YY\end{smallmatrix}\right]$, $\mathcal{M}_3 = \begin{smallmatrix}-\\+\end{smallmatrix}\left[\begin{smallmatrix}YY\\ZZ\end{smallmatrix}\right]$. One obtains $\mathcal{M}_2$ from $\mathcal{M}_1$ by left-multiplying the second row by the first. Similarly, $\mathcal{M}_3$ is obtained from $\mathcal{M}_1$ or $\mathcal{M}_2$ via row multiplication. Observe that, multiplying any row by itself yields II, which stabilizes $|\psi\rangle$. However, II cannot be used as a generator because it is redundant and carries no information about the structure of $|\psi\rangle$. The storage cost for $\mathcal{M}$ is $\Theta(n^2)$, which is an *exponential improvement* over the $O(2^n)$ cost often encountered in vector-based representations.

Stabilizer-circuit simulation. The computational basis states are stabilizer states that can be represented by the stabilizer-matrix structure depicted in Figure 2-a. In this matrix form, the $\pm$ sign of each row along with its corresponding Z_j-literal designates whether the state of the j^{th} qubit is $|0\rangle$ $(+)$ or $|1\rangle$ $(-)$. Suppose we want to simulate circuit $\mathcal{C}$. Stabilizer-based simulation first initializes $\mathcal{M}$ to specify some basis state. Then, to simulate the action of each gate $U \in \mathcal{C}$, we conjugate each row Q_i of $\mathcal{M}$ by U.[7] We require that $UQ_iU^\dagger$ maps to another string of Pauli literals so that the resulting matrix $\mathcal{M}'$ is well-formed. It turns out that the H, P and CNOT gates have such mappings, i.e., these gates conjugate the Pauli group onto itself [7], [11]. Table II lists the mapping for each of these gates. For example, suppose we simulate a CNOT operation on $|\psi\rangle = (|00\rangle + |11\rangle)/\sqrt{2}$. Using the stabilizer representation, we have $\mathcal{M}_\psi = \left[\begin{smallmatrix}+XX\\+ZZ\end{smallmatrix}\right] \xrightarrow{CNOT} \mathcal{M}'_\psi = \left[\begin{smallmatrix}+XI\\+IZ\end{smallmatrix}\right]$. One can verify that the rows of $\mathcal{M}'_\psi$ stabilize $|\psi\rangle \xrightarrow{CNOT} (|00\rangle+|10\rangle)/\sqrt{2}$ as required. Since H, P and CNOT gates are directly simulated using stabilizers, these gates are commonly called *stabilizer gates* and any circuit composed exclusively of such gates is called a unitary *stabilizer circuit*. Table II shows that at most two columns of $\mathcal{M}$ are updated when a stabilizer gate is simulated. Thus, such gates are simulated in $\Theta(n)$ time.

The stabilizer formalism also admits measurements in the computational basis [7]. Conveniently, the formalism avoids the direct computation of projection operators and inner products (Section II). Note that any qubit j in a stabilizer state is either in a $|0\rangle$ $(|1\rangle)$ state or in an unbiased superposition of both. The former case is called a *deterministic outcome* and the latter a *random outcome*.

[7]Since $Q_i |\psi\rangle = |\psi\rangle$, the resulting state $U|\psi\rangle$ is stabilized by $UQ_iU^\dagger$ because $(UQ_iU^\dagger)U|\psi\rangle = UQ_i|\psi\rangle = U|\psi\rangle$.

TABLE II
CONJUGATION OF PAULI-GROUP ELEMENTS BY STABILIZER GATES [11]. FOR $CNOT$, SUBSCRIPT 1 INDICATES THE CONTROL AND 2 THE TARGET.

GATE	INPUT	OUTPUT
H	X	Z
	Y	$-Y$
	Z	X
P	X	Y
	Y	$-X$
	Z	Z

GATE	INPUT	OUTPUT
$CNOT$	I_1X_2	I_1X_2
	X_1I_2	X_1X_2
	I_1Y_2	Z_1Y_2
	Y_1I_2	Y_1X_2
	I_1Z_2	Z_1Z_2
	Z_1I_2	Z_1I_2

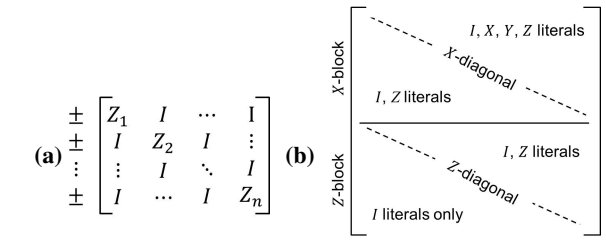

Fig. 2. (a) Stabilizer-matrix structure for basis states. (b) Row-echelon form for stabilizer matrices. The X-block contains a *minimal* set of generators with X/Y literals. Generators with Z and I literals only appear in the Z-block.

We can tell these cases apart in $\Theta(n)$ time by searching for X or Y literals in the j^{th} column of $\mathcal{M}$. If such literals are found, the qubit must be in a superposition and the outcome is random with equal probability ($p(0) = p(1) = .5$); otherwise the outcome is deterministic ($p(0) = 1$ or $p(1) = 1$).

Randomized-outcome case: one flips an unbiased coin to decide the outcome and then updates $\mathcal{M}$ to make it consistent with the outcome obtained. Since we might have to examine $\mathcal{M}$ in its entirety, the runtime is $O(n^2)$.

Deterministic-outcome case: no updates to $\mathcal{M}$ are necessary but we need to figure out whether the qubit is in the $|0\rangle$ or $|1\rangle$ state, i.e., whether the qubit is stabilized by Z or -Z. One approach is to perform Gaussian elimination (GE) to put $\mathcal{M}$ in row-echelon form. This removes redundant literals from $\mathcal{M}$ and makes it possible to identify the row containing a Z in its j^{th} position and I's everywhere else. The $\pm$ phase of such a row decides the outcome of the measurement. Since this is a GE-based approach, it takes $O(n^3)$ time in practice.

The work in [1] improved the runtime of deterministic measurements by doubling the size of $\mathcal{M}$ to include n *destabilizer generators*. Such destabilizer generators help identify exactly which row multiplications to compute in order to decide the measurement outcome. This approach avoids GE and thus deterministic measurements are computed in $O(n^2)$ time.

III. SIMULATION OF QUANTUM CIRCUITS USING STABILIZER FRAMES

The stabilizer gates by themselves do not form a universal set for quantum computation [1], [11]. However, the Hadamard and Toffoli (TOF) gates do [2]. Thus, it suffices to show how to simulate the Toffoli gate using the stabilizer formalism in order to make our gate set universal. To accomplish this, we represent arbitrary quantum states as *superpositions of stabilizer states*. For example, recall from Section II-B that the computational basis states are stabilizer states. Thus, any one-qubit state $|\psi\rangle = \alpha_1|0\rangle + \alpha_2|1\rangle$ is a superposition of the two stabilizer states $|0\rangle$ and $|1\rangle$. Observe that, if $|\psi\rangle$ is *unbiased*, i.e., $|\alpha_1|^2 = |\alpha_2|^2$, it can represented using a single stabilizer state instead of two (up to a global phase). The key idea behind our technique is to identify and compress large unbiased superpositions on the fly during simulation to reduce resource requirements.

Stabilizer frames. Suppose $|\psi\rangle$ is an n-qubit stabilizer state and we want to simulate the action of $TOF_{c_1c_2t}$, where c_1 and c_2 are the control qubits, and t is the target. First, we decompose $|\psi\rangle$ into all four of its double cofactors (Section II) over the control qubits,

$$|\psi\rangle = (|\psi_{c_1c_2=00}\rangle + |\psi_{c_1c_2=01}\rangle + |\psi_{c_1c_2=10}\rangle + |\psi_{c_1c_2=11}\rangle)/2$$

which is an unbiased superposition of orthogonal states. Since $|\psi\rangle$ is a stabilizer state and the cofactors are obtained by performing measurements on $|\psi\rangle$, each $|\psi_{c_1c_2}\rangle$ is computed in $O(n^2)$ time (Section II-B). We compute the action of the Toffoli as,

$$TOF_{c_1c_2t}|\psi\rangle = (|\psi_{c_1c_2=00}\rangle + |\psi_{c_1c_2=01}\rangle + |\psi_{c_1c_2=10}\rangle + X_t|\psi_{c_1c_2=11}\rangle)/2$$

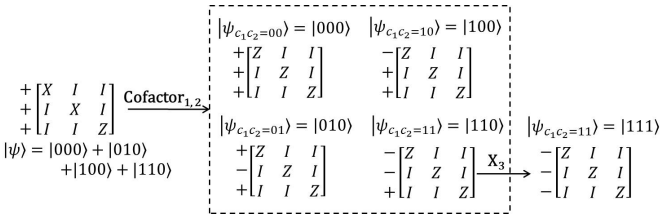

Fig. 3. Simulation of the Toffoli gate using a superposition of stabilizer states. Amplitudes are omitted for clarity. The X gate is applied to the third qubit of the $|\psi_{c_1 c_2=11}\rangle$ cofactor. The ($\pm$)-phase vectors are shown as prepended columns to the corresponding stabilizer matrices.

where X_t is the Pauli gate (NOT) acting on target t. Each $|\psi_{c_1 c_2}\rangle$ is represented by the same $\mathcal{M}$, but with a different permutation of leading row phases as shown in Figure 3. Thus, one can represent the orthogonal stabilizer-state superpositions that arise when simulating Toffoli gates by a *stabilizer frame* $\mathcal{F}$ consisting of (*i*) a stabilizer matrix $\mathcal{M}$ and (*ii*) a set of k distinct leading ($\pm$)-phase vectors. Each phase vector in the frame represents a distinct state in the superposition. Additionally, one maintains a vector $\mathbf{a} = (a_1, \ldots, a_k)$ of the amplitudes associated with the states (phase vectors) in the superposition, e.g., $\mathbf{a} = (.5, .5, .5, .5)$ in Figure 3. Controlled-phase gates $R(\alpha)_{ct}$ can also be simulated using stabilizer frames. This gate applies a phase-shift factor of $e^{i\alpha}$ if both the control qubit c and target qubit t are set. Thus, we compute the action of $R(\alpha)_{ct}$ as,

$$R(\alpha)_{ct}\,|\psi\rangle = (\,|\psi_{ct=00}\rangle + |\psi_{ct=01}\rangle + |\psi_{ct=10}\rangle + e^{i\alpha}\,|\psi_{ct=11}\rangle)/2$$

Observe that, in contrast to TOF gates, controlled-$R(\alpha)$ gates produce biased superpositions. The Hadamard and controlled-$R(\alpha)$ gates are used to implement the quantum Fourier transform circuit, which plays a key role in Shor's factoring algorithm.

A. Frame-based Simulation

We now discuss how to manipulate a stabilizer frame $\mathcal{F}$ in order to simulate generic quantum circuits with both stabilizer and non-stabilizer gates. To simulate stabilizer gates, we first update the stabilizer matrix $\mathcal{M}$ associated with $\mathcal{F}$ as per Section II-B. Then, we iterate over the phase vectors in $\mathcal{F}$ and update each accordingly (Table II). Thus, this operation takes $O(nk)$ time for a superposition with k states. To simulate a non-stabilizer gate, we first update $\mathcal{M}$ (i.e., apply measurements to obtain relevant cofactors). We then iterate over each phase vector in $\mathcal{F}$ and permute the corresponding phases in order to generate additional phase vectors corresponding to the cofactor states. As in the case of stabilizer gates, this operation is linear in the number of phase vectors. However, by the end of the operation, the number of phase vectors (states) in $\mathcal{F}$ will have grown by a (worst case) factor of four in the case of both TOF and controlled-$R(\alpha)$. For an arbitrary n-qubit stabilizer frame $\mathcal{F}$, the number of phase vectors is upper bounded by 2^n, the number of possible $\pm$ permutations.

Prior work on simulation of non-stabilizer gates using the stabilizer formalism can be found in [1] where the authors propose an approach that represents a quantum state as a sum of $O(4^{2d})$ density-matrix terms, where d is the number of distinct qubits involved in non-stabilizer operations.

Global phases of states in $\mathcal{F}$. In quantum mechanics, the states $e^{i\theta}\,|\psi\rangle$ and $|\psi\rangle$ are considered phase-equivalent because $e^{i\theta}$ does not affect the statistics of measurement. During stabilizer-based simulation, such global phases are not maintained. Since these phases are unobservable, this is not a problem when simulating a single stabilizer state. However, since we manipulate superpositions of states, such global phases become relative and cannot be ignored. In frame-based simulation, we maintain the global phases of the states in $\mathcal{F}$ using the amplitude vector $\mathbf{a}$. Let $\mathbf{p_i}$ be the phase-vector associated with $a_i \in \mathbf{a}$. When simulating gate U, we update each a_i as follows:

1) Set the leading phases of the rows in $\mathcal{M}$ to $\mathbf{p_i}$.
2) Obtain a basis state $|b\rangle$ from $\mathcal{M}$ and store its amplitude β. If U is the Hadamard gate, it may be necessary to sample a sum of two non-zero basis amplitudes (one real, one imaginary).
3) Compute $U(\beta\,|b\rangle) = \beta'\,|b'\rangle$ via the state-vector representation.
4) Obtain $|b'\rangle$ from $U\mathcal{M}U^\dagger$ and store its non-zero amplitude γ.
5) Compute the global-phase factor generated as $a_i = (a_i \cdot \beta')/\gamma$.

To sample the computational-basis amplitudes $|b\rangle$ and $|b'\rangle$ from the stabilizer, $\mathcal{M}$ needs to be in row-echelon form (Figure 2-b). Thus, each global-phase computation takes $O(n^3)$ time for an n-qubit $\mathcal{M}$. To improve this, we introduce a simulation invariant.

Invariant 1: The stabilizer matrix $\mathcal{M}$ associated with $\mathcal{F}$ remains in row-echelon form (Figure 2b) during simulation.

Since stabilizer gates affect at most two columns of $\mathcal{M}$, Invariant 1 can be repaired with $O(n)$ row multiplications. Since each row multiplication takes $\Theta(n)$, the runtime required to update $\mathcal{M}$ during global-phase maintenance simulation is $O(n^2)$. Therefore, for an arbitrary n-qubit stabilizer frame with k states, the overall runtime for simulating a single gate is $O(n^2 + nk)$ since one can memoize the updates to $\mathcal{M}$ required to compute each a_i.

Measuring $\mathcal{F}$. Since the states in $\mathcal{F}$ are orthogonal, the outcome probability when measuring $\mathcal{F}$ is calculated as the sum of the normalized outcome probabilities of each state. The normalization is with respect to the amplitudes stored in $\mathbf{a}$ and thus the overall measurement outcome may have a non-uniform distribution. Formally, let $\Psi = \sum_i a_i\,|\psi_i\rangle$ be the superposition of states represented by $\mathcal{F}$, the probability of observing outcome $x \in \{0, 1\}$ upon measuring qubit m is,

$$p(x)_\Psi = \sum_{i=1}^{k} |a_i|^2\,\langle\psi_i|\,P_x^m\,|\psi_i\rangle = \sum_{i=1}^{k} |a_i|^2 p(x)_{\psi_i}$$

where P_x^m denotes the projection operator in the computational basis x as discussed in Section II. The outcome probability for each stabilizer state $p(x)_{\psi_i}$ is computed as outlined in Section II-B. Once we compute $p(x)_\Psi$, we flip a (possibly biased) coin to decide the outcome and update the stabilizer matrix associated with $\mathcal{F}$ (Section II-B). In the worst case, the outcomes of all the states in Ψ are random and each require an $O(n^2)$-time update to $\mathcal{M}$. (Deterministic measurements do not require updates to $\mathcal{M}$ and, since we maintain Invariant 1, such measurements can be decided in linear time.) Thus, measuring a frame with k states takes $O(n^2 + nk)$ time.

Multiframe simulation. Although a single frame is sufficient to represent a stabilizer-state superposition Ψ, one can tame the exponential growth of states in Ψ by admitting a multiframe representation. Such a representation cuts down the total number of states required to represent Ψ by at least a half, thus improving the scalability of our technique. Our experiments in Section IV show that, when simulating ripple-carry adders, the number of states in Ψ grows linearly when multiframes are used but exponentially when a single frame is used.

One derives a multiframe representation directly from a single frame $\mathcal{F}$ by examining the set of phase vectors and identifying *candidate pairs* that can be *coalesced* into a single phase vector associated with a different stabilizer matrix. Since we maintain the stabilizer matrix $\mathcal{M}$ of a frame in row-echelon form (Invariant 1), examining the phases corresponding to Z_j rows (Z-literal in j^{th} column and I's in all other columns) allows us to identify the columns in $\mathcal{M}$ that need to be modified in order to coalesce candidate pairs. Figure 4 shows an example of this process. To obtain $\mathcal{M}_1$ in the Figure 4 example, we conjugate the first column of $\mathcal{M}$ by an H gate. Similarly, to obtain $\mathcal{M}_2$ we conjugate the first column by H and then conjugate the first and third columns by CNOT. Thus, the output of this coalescing process is a list of frames $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_l$

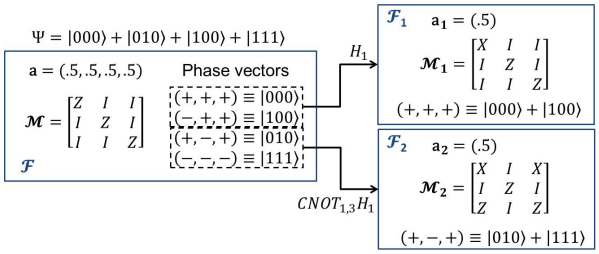

Fig. 4. Example of how a multiframe representation is derived from a single-frame representation. Each frame $\mathcal{F}_i$ consists of a stabilizer matrix $\mathcal{M}_i$, a set of ($\pm$)-phase vectors and a vector of amplitudes $\mathbf{a_i}$.

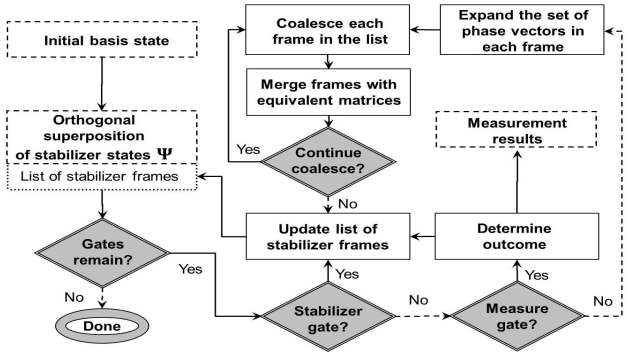

Fig. 5. Simulation flow for Quipu.

that together represent the same superposition as the original input frame. We introduce the following invariant to facilitate simulation of quantum measurements on multiple frames.

Invariant 2: The stabilizer frames that represent a superposition of stabilizer states remain mutually orthogonal during simulation, i.e., every pair of (basis) vectors from any two frames are orthogonal.

To maintain Invariant 2 we define a specific type of candidate pair such that the new frames generated from the set of coalesced phase vectors are mutually orthogonal. Suppose $\langle \mathbf{p}_r, \mathbf{p}_j \rangle$ are a pair of phase vectors from the same n-qubit frame. Then $\langle \mathbf{p}_r, \mathbf{p}_j \rangle$ is considered a candidate iff it has the following properties: (*i*) $\mathbf{p}_r$ and $\mathbf{p}_j$ are equal up to $m \leq n$ entries corresponding to Z_k-rows (where k is the qubit the row stabilizes), and (*ii*) $a_r = i^d a_j$ for some $d \in \{0, 1, 2, 3\}$ (where a_r and a_j are the frame amplitudes paired with $\mathbf{p}_r$ and $\mathbf{p}_j$). The stabilizer circuit needed to coalesce a candidate pair is defined as $\mathcal{C}$=CNOT$_{v_1,v_2}$CNOT$_{v_1,v_3} \cdots$ CNOT$_{v_1,v_m}$P$^d_{v_1}$H$_{v_1}$, where the v_k designate the qubits stabilized by the m differing entries in the candidate pair. The steps in our coalescing procedure are:

1) Sort phase vectors according to differing entries such that *candidate pairs* are next to each other.
2) Coalesce candidate pairs into a new set of phase vectors.
3) Create a new frame $\mathcal{F}_i$ consisting of the set of coalesced phase vectors and the new stabilizer matrix $\mathcal{C}\mathcal{M}\mathcal{C}^\dagger$.
4) Repeat steps 2–3 until no candidate pairs remain.

The runtime of this procedure is dominated by Step 1. Each phase-vector comparison takes $\Theta(n)$ time, where n is the size of the phase vectors. Therefore, the runtime of step 1 and our overall coalescing procedure is $O(nk \log k)$ for a single frame with k phase vectors.

To simulate stabilizer, TOF, controlled-$R(\alpha)$ and measurement gates using multiple frames, one applies our single-frame algorithms to each frame in the list independently. In the case of TOF and controlled-$R(\alpha)$ gates, additional steps are required:

1) Apply the coalescing procedure to each frame and insert the new "coalesced" frames in the list.
2) Merge frames with equivalent stabilizer matrices.
3) Repeat Steps 1 and 2 until no new frames are generated.

The simulation flow of our technique is shown in Figure 5 and implemented in our software package Quipu.

IV. Empirical Validation

We tested a single-threaded version of Quipu on a conventional Linux server using several benchmark sets consisting of stabilizer circuits, quantum ripple-carry adders, quantum Fourier transform circuits and quantum fault-tolerant (FT) circuits.

Stabilizer circuits. We compared the runtime performance of Quipu against that of CHP using a benchmark set similar to the one used in [1]. We generated random stabilizer circuits on n qubits, for $n \in \{100, 200, \ldots, 1500\}$. The use of randomly generated benchmarks is justified for our experiments because (*i*) our algorithms are not explicitly sensitive to circuit topology and (*ii*) random stabilizer circuits have been considered representative [9]. For each n, we generated the circuits as follows: fix a parameter $\beta > 0$; then choose $\beta\lceil n \log_2 n \rceil$ random unitary gates (CNOT, P or H) each with probability $1/3$. Then measure each qubit $a \in \{0, \ldots, n-1\}$ in sequence. We measured the number of seconds needed to simulate the entire circuit. The entire procedure was repeated for β ranging from 0.6 to 1.2 in increments of 0.1. Figure 6 shows the average time needed by Quipu and CHP to simulate this benchmark set. The purpose of this comparison is to evaluate the overhead of supporting generic circuit simulation in Quipu. Since CHP is specialized to stabilizer circuits, we do not expect Quipu to be faster. When $\beta = 0.6$, the simulation time appears to grow roughly linearly in n for both simulators. However, when the number of unitary gates is doubled ($\beta = 1.2$), the runtime of both simulators grows roughly quadratically. Thus, the performance of both CHP and Quipu depends strongly on the circuit being simulated. Although Quipu is $5\times$ slower than CHP, we note that Quipu maintains global phases whereas CHP does not. Figure 6 shows that Quipu is asymptotically as fast as CHP when simulating stabilizer circuits that contain a linear number of measurements.

Ripple-carry adders. Our second benchmark set consists of n-bit ripple-carry (Cuccaro) adder [4] circuits, which often appear as components in many arithmetic circuits [10]. The Cuccaro circuit for $n = 3$ is shown in Figure 7. Such circuits act on two n-qubit input registers, one ancilla qubit and one carry qubit for a total of $2(n+1)$ qubits. We applied H gates to all $2n$ input qubits in order to simulate addition on a superposition of 2^{2n} computational-basis states. Figure 8 shows the average runtime needed to simulate this benchmark set using Quipu. For comparison, we ran the same benchmarks on an optimized version of QuIDDPro, called QPLite[8], specific to circuit simulation [18]. When $n < 15$, QPLite

[8] QPLite is up to $4\times$ faster since it removes overhead related to QuIDDPro's interpreted front-end for extended quantum programming [15].

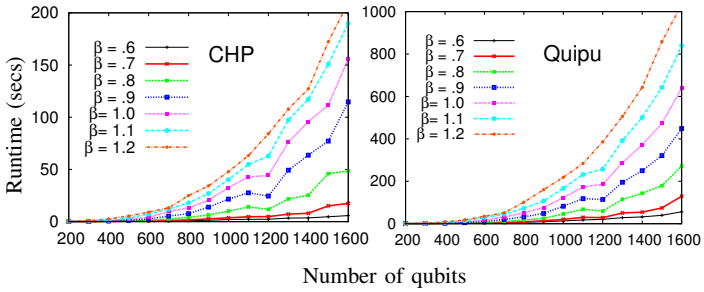

Fig. 6. Average time needed by Quipu and CHP to simulate an n-qubit stabilizer circuit with $\beta n \log n$ gates and n measurements. Quipu is asymptotically as fast as CHP but is not limited to stabilizer circuits.

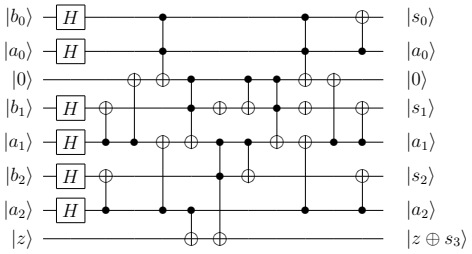

Fig. 7. Ripple-carry (Cuccaro) adder for 3-bit numbers $a = a_0a_1a_2$ and $b = b_0b_1b_2$. The third qubit from the top is an ancilla and the z qubit is the carry. The b-register is overwritten with the result $s_0s_1s_2$.

is faster than Quipu because the QuIDD representing the state vector remains compact during simulation. However, for $n > 15$, the compactness of the QuIDD is considerably reduced, and the majority of QPLite's runtime is spent in non-local pointer-chasing and memory (de)allocation. Thus, QPLite fails to scale on such benchmarks and one observes an exponential increase in runtime. Furthermore, Quipu consumed 62% less memory than QPLite in each of these benchmarks.

We ran the same benchmarks using both the single-frame and multiframe approaches. In the case of a single frame, the number of states in a superposition grows exponentially in n. However, in the multiframe approach, the number of states grows linearly in n. This is because TOF gates produce large equal superpositions that are effectively compressed by our coalescing technique. Since our frame-based algorithms require poly(k) time for k states in a superposition, Quipu simulates Cuccaro circuits in polynomial time and space for input states consisting of large superpositions of basis states. On such instances, known linear-algebraic simulation techniques (e.g., QuIDDPro) take exponential time.

The work in [10] describes additional quantum arithmetic circuits that are based on Cuccaro adders (e.g., subtractors, conditional adders, comparators). We used Quipu to simulate such circuits and observed similar runtime performance as that shown in Figure 8.

Quantum Fourier transform (QFT) circuits. Our third benchmark set consists of circuits for implementing the n-qubit QFT, which computes the discrete Fourier transform of the amplitudes in the input quantum state. Let $|x_1x_2\ldots x_n\rangle$, $x_i \in \{0,1\}$ be a computational basis state and $\mathbf{x}_{1,2,\ldots,m} = \sum_{k=1}^{m} x_k 2^{-k}$. The action of the QFT on input state can be expressed as:

$$|x_1\ldots x_n\rangle = \frac{1}{\sqrt{2^n}}\left(|0\rangle + e^{2i\pi\cdot\mathbf{x}_n}|1\rangle\right)\otimes\left(|0\rangle + e^{2i\pi\cdot\mathbf{x}_{n-1,n}}|1\rangle\right)\otimes \cdots\otimes\left(|0\rangle + e^{2i\pi\cdot\mathbf{x}_{1,2,\ldots,n}}|1\rangle\right) \quad (1)$$

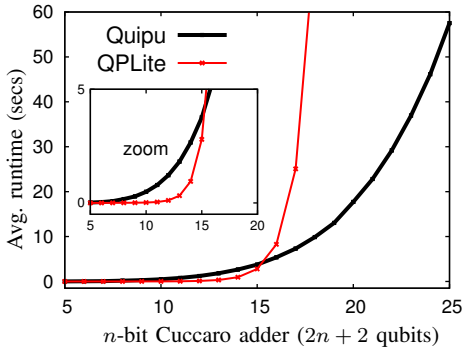

Fig. 8. Average runtime needed by Quipu and QuIDDPro to simulate n-bit Cuccaro adders after an equal superposition of allcomputational basis states is obtained using a block of Hadamard gates (Figure 7). Quipu consumed 62% less memory than QPLite for each of these benchmarks.

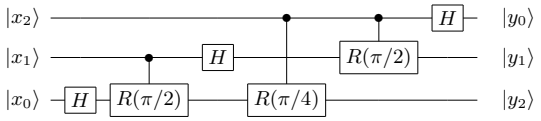

Fig. 9. The three-qubit QFT circuit. In general, The first qubit requires one Hadamard gate, the next qubit requires a Hadamard and a controlled-$R(\alpha)$ gate, and each following qubit requires an additional controlled-$R(\alpha)$ gate. Summing up the number of gates gives $O(n^2)$ for an n-qubit QFT circuit.

The QFT is used in many quantum algorithms, notably Shor's factoring and discrete logarithm algorithms. Such circuits are composed of a network of Hadamard and controlled-$R(\alpha)$ gates, where $\alpha = \pi/2^k$ and k is the distance over which the gate acts. The three-qubit QFT circuit is shown in Figure 9. Figure 10 shows average runtime and memory usage for both Quipu and QPLite on QFT instances for $n = \{10, 12, \ldots, 20\}$. Quipu runs approximately $4\times$ faster than QPLite on average and consumes about 90% less memory. For these benchmarks, we observed that the number of states in our multiframe data structure was 2^{n-1}. This is because controlled-$R(\alpha)$ gates produce biased superpositions (Section III-A) that cannot be effectively compressed using our coalescing procedure. Therefore, as Figure 10 shows, the runtime and memory requirements of both Quipu and QPLite grow exponentially in n for QFT instances. However, Quipu scales to 22-qubit instances whereas QPLite scales to only 18 qubits.

Fault-tolerant (FT) circuits. Our last benchmark set consists of circuits that, in addition to preparing encoded quantum states, implement procedures for performing FT quantum operations [6], [11], [14]. FT operations limit the propagation errors from one qubit in a QECC-register (the block of qubits that encodes a logical qubit) to another qubit in the same register, and a single faulty gate damages at most one qubit in each register. One constructs FT stabilizer circuits by executing each stabilizer gate transversally[9] across QECC-registers [7], [11], [14]. Non-stabilizer gates need to be implemented using a FT architecture that often requires additional ancilla qubits, measurements and correction procedures conditioned on measurement outcomes. Figure 11 shows a circuit that implements a FT-Toffoli operation [14]. Each line in Figure 11 represents a 5-qubit register implementing the DiVincenzo/Shor code.

We implemented FT benchmarks for the half-adder and full-adder circuits as well as for computing $f(x) = b^x \bmod 15$. Each circuit from Figure 12 implements $f(x)$ with a particular co-prime base value b as a $(2,4)$ look-up table (LUT).[10] The Toffoli gates in all our FT benchmarks are implemented using the FT architecture from Figure 11. Since FT-Toffoli operations require 6 ancilla registers, a circuit

[9] In a transversal operation, the i^{th} qubit in each QECC-register interacts only with the i^{th} qubit of other QECC-registers.

[10] A (k,m)-LUT takes k read-only input bits and $m > \log_2 k$ ancilla bits. For each 2^k input combination, an LUT produces a pre-determined m-bit value, e.g., a $(2,4)$-LUT is defined by values $(1,2,4,8)$ or $(1,4,1,4)$.

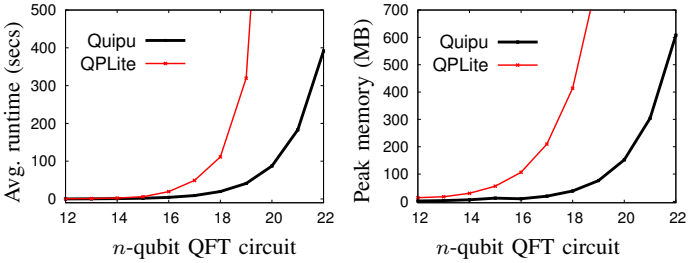

Fig. 10. Average runtime and memory needed by Quipu and QuIDDPro to simulate n-qubit QFT circuits, which contain $n(n+1)/2$ gates.

TABLE III

AVERAGE TIME AND MEMORY NEEDED BY QUIPU AND QPLITE TO SIMULATE OUR BENCHMARK SET OF QUANTUM FT CIRCUITS. THE SECOND COLUMN INDICATES THE QECC USED TO ENCODE k LOGICAL QUBITS INTO n PHYSICAL QUBITS. WE USED THE 3-QUBIT BIT-FLIP CODE FOR LARGER BENCHMARKS AND THE 5-QUBIT DIVINCENZO/SHOR CODE [6] FOR SMALLER ONES (*).

FAULT-TOLERANT CIRCUIT	QECC $[n,k]$	TOTAL QUBITS (INC. ANCILLA)	NUM. OF GATES		RUNTIME (SECS)		MEMORY (MB)		MAX SIZE(Ψ)	
			STAB.	TOFF.	QPLite	Quipu	QPLite	Quipu	SINGLE $\mathcal{F}$	MULTI $\mathcal{F}$
toffoli*	$[15,3]$	45	155	15	43.68	**0.20**	98.45	**12.76**	2816	32
halfadd*	$[15,3]$	45	160	15	43.80	**0.20**	94.82	**12.76**	2816	32
fulladd*	$[20,4]$	80	320	30	84.96	**0.88**	91.86	**12.94**	2816	32
2^xmod15	$[18,6]$	81	396	36	4.81hrs	**1.48**	11.85	**12.96**	22528	64
4^xmod15*	$[30,6]$	30	30	0	0.01	$<$ **0.01**	6.14	**12.01**	1	1
7^xmod15	$[18,6]$	81	402	36	11.25hrs	**1.52**	12.41	**13.29**	22528	64
8^xmod15	$[18,6]$	81	399	36	11.37hrs	**1.52**	12.48	**13.29**	22528	64
11^xmod15*	$[30,6]$	30	25	0	0.02	$<$ **0.01**	6.14	**12.01**	1	1
13^xmod15	$[18,6]$	81	399	36	11.28hrs	**1.56**	11.85	**12.25**	22528	64
14^xmod15*	$[30,6]$	30	40	0	0.02	$<$ **0.01**	6.14	**12.01**	1	1

that implements t FT-Toffolis using a k-qubit QECC, requires $6tk$ ancilla qubits. Therefore, to compare with QPLite, we used the 3-qubit bit-flip code [11, Ch. 10] instead of the more robust 5-qubit code in our larger benchmarks. Our results in Table III show that Quipu is typically faster than QPLite by several orders of magnitude and consumes $8\times$ less memory for the *toffoli*, *half-adder* and *full-adder* benchmarks. Table III also shows that our coalescing technique is effective as the maximum size of the stabilizer-state superposition is orders-of-magnitude smaller when multiple frames are used.

V. CONCLUSIONS AND FUTURE WORK

In this work, we developed new techniques for quantum-circuit simulation based on superpositions of stabilizer states, and managed to circumvent shortcomings in prior work [1]. We implement our algorithms in our software package Quipu. Current simulators based on the stabilizer formalism, such as CHP, are limited to simulation of stabilizer circuits. Our results show that Quipu performs asymptotically as fast as CHP on stabilizer circuits with a linear number of measurement gates. Our stabilizer-based technique simulates certain quantum arithmetic circuits in polynomial time and space for input states consisting of unbiased superpositions of computational-basis states. QuIDDPro takes exponential time on such instances. We simulated various quantum Fourier transform and quantum fault-tolerant circuits with Quipu, and the results demonstrate that our stabilizer-based technique leads to orders-of-magnitude improvement in runtime and memory as compared to QuIDDPro. While our technique uses more sophisticated mathematics and quantum-state modeling, it is significantly easier to implement and optimize.

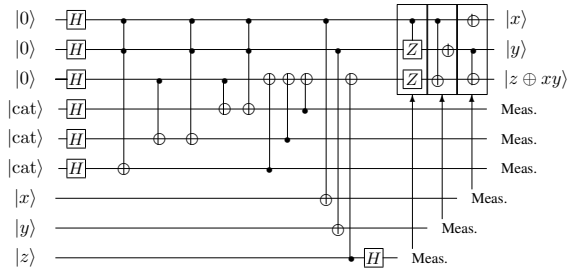

Fig. 11. Fault-tolerant implementation of a Toffoli gate. Each line represents a 5-qubit register and each gate is applied transversally. The state $|cat\rangle = (|0^{\otimes 5}\rangle + |1^{\otimes 5}\rangle)/\sqrt{2}$ is obtained using a stabilizer subcircuit (not shown). The arrows point to the set of gates that is applied if the measurement outcome is 1; no action is taken otherwise. Controlled-Z gates are implemented as $H_j CNOT_{i,j} H_j$ with control i and target j. Z gates are implemented as P^2.

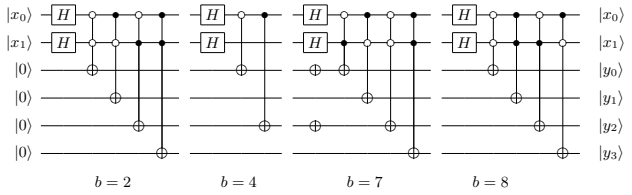

Fig. 12. Mod-exp with $M = 15$ implemented as $(2,4)$-LUTs [10] for several co-prime base values. Negative controls are shown with hollow circles. We apply Hadamards to each x-qubit to generate a superposition of all the input values for x. Our benchmarks implement these computations using the 3-qubit bit-flip code [11, Ch. 10] and the FT-Toffoli architecture from Figure 11.

REFERENCES

[1] S. Aaronson, D. Gottesman, "Improved Simulation of Stabilizer Circuits," *Phys. Rev. A*, vol. 70, no. 052328 (2004).

[2] D. Aharonov. "A Simple Proof that Toffoli and Hadamard are Quantum Universal," arXiv:quant-ph/0301040 (2003).

[3] O. Boncalo et al., "Using Simulated Fault Injection for Fault Tolerance Assessment of Quantum Circuits," *Proc. Sim. Symp.*, pp.213-220 (2007).

[4] S. A. Cuccaro et al., "A New Quantum Ripple-carry Addition Circuit," arXiv:quant-ph/0410184v1 (2004).

[5] K. De Raedt et al., "Massively Parallel Quantum Computer Simulator", *Comp. Phys. Comm.*, vol. 176, no. 2, pp. 121–136 (2007).

[6] D. P. DiVincenzo, P. W. Shor, "Fault-Tolerant Error Correction with Efficient Quantum Codes", *Phys. Rev. Lett.*, vol. 77, no. 3260 (1996).

[7] D. Gottesman, "The Heisenberg Representation of Quantum Computers," arXiv:9807006v1 (1998).

[8] L. Grover, "A Fast Quantum Mechanical Algorithm for Database Search," *Symp. on Theory of Comp.*, pp. 212-219 (1996).

[9] E. Knill et al., "Randomized Benchmarking of Quantum Gates," *Phys. Rev. A*, vol. 77, no. 1 (2007).

[10] I. L. Markov, M. Saeedi, "Constant-optimized Quantum circuits for Modular Multiplication and Exponentiation," *Quant. Info. and Comp.*, vol. 12, no. 5 (2012).

[11] M. A. Nielsen, I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press (2000).

[12] K. M. Obenland, A. M. Despain, "A Parallel Quantum Computer Simulator, " arXiv:quant-ph/9804039 (1998).

[13] B. Oemer (2003), http://tph.tuwien.ac.at/~oemer/qcl.html.

[14] J. Preskill, "Fault Tolerant Quantum Computation," *Introduction to Quantum Computation*, World Scientific (1998). quant-ph/9712048.

[15] http://vlsicad.eecs.umich.edu/Quantum/qp/

[16] V. V. Shende, S. S. Bullock, I. L. Markov, "Synthesis of quantum logic circuits," *IEEE Trans. on CAD*, vol. 25, no. 6 (2006).

[17] P. Shor, "Polynomial-time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM J. Comput*, vol. 26, no. 5 (1997).

[18] G. F. Viamontes, I. L. Markov, J. P. Hayes, *Quantum Circuit Simulation*, Springer (2009).