Simulation of Quantum Circuits via Stabilizer Frames

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

Abstract—Generic quantum-circuit simulation appears intractable for conventional computers and may be unnecessary because useful quantum circuits exhibit significant structure that can be exploited during simulation. For example, Gottesman and Knill identified an important subclass, called stabilizer circuits, which can be simulated efficiently using group-theory techniques and insights from quantum physics. Realistic circuits enriched with quantum error-correcting codes and fault-tolerant procedures are dominated by stabilizer subcircuits and contain a relatively small number of non-Clifford components. Therefore, we develop new data structures and algorithms that facilitate parallel simulation of such circuits. Stabilizer frames offer more compact storage than previous approaches but require more sophisticated bookkeeping. Our implementation, called Quipu, simulates certain quantum arithmetic circuits (e.g., reversible 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 quantum Fourier transform and quantum fault-tolerant circuits using Quipu, and the results demonstrate that our stabilizer-based technique empirically outperforms QuIDDPro in all cases. While previous high-performance, structure-aware simulations of quantum circuits were difficult to parallelize, we demonstrate that Quipu can be parallelized with a nontrivial computational speedup.

1 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, optics) and used in recently developed commercial products. Examples of such products include MagiQ’s quantum key distribution system and ID-Quantique’s quantum random number generator. Shor’s factoring algorithm [24] and Grover’s search algorithm [19] 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. For instance, the work in [22] describes practical FT architectures for quantum computers, and [15] explores architectures for constructing reliable communication channels via distribution of high-fidelity EPR pairs in a large quantum computer. 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. In particular, high-performance simulation is a key component in quantum design flows [25] that facilitates analysis of trade-offs between performance and accuracy. 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) [21] and Viamontes’ Quantum Information Decision Diagrams (QuIDD) implemented in the QuIDDPro package [26]. 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. fault-injection analysis) of quantum FT architectures using general-purpose simulators has been limited to small quantum circuits [5]. Therefore, designing fast simulation techniques that target quantum FT circuits facilitates more robust reliability analysis of larger quantum circuits.

Stabilizer circuits and states. Gottesman [12] 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 Clifford gates – Hadamard, Phase and controlled-NOT gates (Figure 1) followed by one-qubit measurements in the computational basis. Such cir-

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

Fig. 1. Clifford gates: Hadamard (H), Phase (P) and controlled-NOT (CNOT).
cuits are applied to a computational basis state (usually \(|00\ldots0\rangle\)) 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], [2]. Stabilizer circuits can be simulated in polynomial-time by keeping track of the Pauli operators that stabilize the quantum state. Such stabilizer operators are maintained during simulation and uniquely represent stabilizer states up to an unobservable global phase [3]. Thus, this technique offers an exponential improvement over the computational resources needed to simulate stabilizer circuits using vector-based representations.

Aaronson and Gottesman [3] proposed an improved technique that uses a bit-vector representation to simulate stabilizer circuits. Aaronson implemented this simulation approach in his CHF software package. Compared to other vector-based simulators (QuIDDPro, QCL) the technique in [3] does not maintain the global phase of a state and simulates each Clifford gate in \(\Theta(n)\) time using \(\Theta(n^2)\) space. The overall runtime of CHF is dominated by measurements, 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-Clifford gates such as Toffoli 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 terms. In contrast, we store arbitrary states as superpositions of pure stabilizer states. Such superpositions are stored more compactly than the approach from [1], although we do not handle mixed stabilizer states. The key obstacle to the more efficient pure-state approach has been the need to maintain the global phase of each stabilizer state in a superposition, where such phases become relative. We develop a new algorithm to overcome this obstacle. 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. **Stabilizer subcircuits are simulated with high efficiency.**
2. **Superpositions can be restructured and compressed on the fly during simulation to reduce resource requirements.**
3. **Operations performed on such superpositions can be computed in parallel and lend themselves to distributed or asynchronous processing.**

Our stabilizer-based technique simulates certain quantum arithmetic circuits in polynomial time and space for input states consisting of equal superpositions of computational-basis states. On such instances, well-known generic simulation techniques take exponential time. We simulate various quantum Fourier transform and quantum fault-tolerant 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 at least a superficial familiarity with quantum computing. Section 2 describes key concepts related to quantum-circuit simulation and the stabilizer formalism. In Section 3 we introduce stabilizer frames and describe relevant algorithms. Section 4 describes in detail our simulation flow implemented in Quipu, and Section 5 discusses a parallel implementation of our technique. In Section 6 we describe empirical validation of Quipu and in single- and multi-threaded variants, as well as comparisons with state-of-the-art simulators. Section 7 closes with concluding remarks.

### 2 Background and Previous Work

Quantum information processes, including quantum algorithms, are often modeled using quantum circuits and, just like conventional digital circuits, are represented by diagrams [19], [26]. Quantum circuits are sequences of **gate operations** that act on some register of **qubits** — the basic unit of information in a quantum system. The quantum state \(|\psi\rangle\) of a single qubit is described by a two-dimensional complex-valued vector. In contrast to classical bits, qubits can be in a superposition of the 0 and 1 states. Formally, \(|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle\), where \(|0\rangle = (1,0)^T\) and \(|1\rangle = (0,1)^T\) are the two-dimensional computational basis states and \(\alpha_i\) 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 \(X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\).

\[
\begin{align*}
\alpha_0 |00\rangle &+ \alpha_1 |10\rangle \xrightarrow{X \otimes I} \alpha_0 |10\rangle + \alpha_1 |00\rangle \\
\end{align*}
\]

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

\[
\begin{align*}
\alpha_0 |00\rangle + \alpha_1 |10\rangle \xrightarrow{CNOT} \alpha_0 |00\rangle + \alpha_1 |11\rangle \\
\end{align*}
\]

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

\[
\begin{align*}
\alpha_0 |00\rangle + \alpha_1 |10\rangle \xrightarrow{H} \alpha_0 (|00\rangle + |01\rangle) + \alpha_1 (|10\rangle + |11\rangle) / \sqrt{2} \\
\end{align*}
\]

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 [19].
5. A superposition is a norm-1 linear combination of terms.
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 a quantum state are described by non-unitary measurement operators [19, Section 2.2.3]. There are different types of quantum measurements, but the type most pertinent to our discussion comprises projective measurements in the computational basis, i.e., measurements with respect to the \( |0 \rangle \) or \( |1 \rangle \) basis states. The corresponding measurement operators are \( P_0 = (\frac{1}{\sqrt{2}} |0 \rangle |0 \rangle + |1 \rangle |1 \rangle) \) and \( P_1 = (\frac{1}{\sqrt{2}} |0 \rangle |1 \rangle - |1 \rangle |0 \rangle) \), 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 \( \langle \psi | P_j | \psi \rangle \), where \( |\psi \rangle \) is the conjugate transpose of \( |\psi \rangle \).

The dynamics involved in observing a quantum state are described by non-unitary measurement operators [19, Section 2.2.3]. There are different types of quantum measurements, but the type most pertinent to our discussion comprises projective measurements in the computational basis, i.e., measurements with respect to the \( |0 \rangle \) or \( |1 \rangle \) basis states. The corresponding measurement operators are \( P_0 = (\frac{1}{\sqrt{2}} |0 \rangle |0 \rangle + |1 \rangle |1 \rangle) \) and \( P_1 = (\frac{1}{\sqrt{2}} |0 \rangle |1 \rangle - |1 \rangle |0 \rangle) \), 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 \( \langle \psi | P_j | \psi \rangle \), where \( |\psi \rangle \) is the conjugate transpose of \( |\psi \rangle \).

2.1 Quantum circuits and simulation

To simulate a quantum circuit \( C \), 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 \( C \) are applied one at a time, eventually producing the output state \( C |\psi \rangle \) [1, 19, 26]. Most quantum-circuit simulators [8, 20, 21, 26] support some form of the linear-algebraic operations described earlier. The drawback of such simulators is that their runtime and memory requirements grow exponentially in the number of qubits. This holds true not only in the worst case but also in practical applications involving quantum arithmetic and quantum FT circuits.

Gottesman developed a simulation method involving the Heisenberg model [12] often used by physicists to describe atomic-scale 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-valued vectors and amplitudes. The symmetries are operators for which these states are 1-eigenvectors. Algebraically, symmetries form group structures, which can be specified compactly by group generators [14].

2.2 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: \( X = (\frac{1}{\sqrt{2}} |0 \rangle |1 \rangle + |1 \rangle |0 \rangle) \), \( Y = (\frac{1}{\sqrt{2}} |0 \rangle i |1 \rangle - |1 \rangle i |0 \rangle) \), and the identity \( I = (\frac{1}{\sqrt{2}} |0 \rangle |0 \rangle + |1 \rangle |1 \rangle) \). The one-qubit stabilizers defined by the Pauli matrices are: \( X = (\frac{1}{\sqrt{2}} |0 \rangle + |1 \rangle)/\sqrt{2} \), \( Y = (\frac{1}{\sqrt{2}} |0 \rangle + i |1 \rangle)/\sqrt{2} \), \( Z = \frac{1}{\sqrt{2}} |0 \rangle - i |1 \rangle \).

Observe that \( I \) stabilizes all states and \( -I \) does not stabilize any state. Thus, the entangled state \( (\frac{1}{\sqrt{2}} |0 \rangle + |1 \rangle)/\sqrt{2} \) is stabilizable by the Pauli operators \( X \otimes X, Y \otimes Y, Z \otimes Z \) and \( I \otimes I \). As shown in Table 1, it turns out that the Pauli matrices along with \( I \) and the multiplicative factors \( \pm 1 \) form a closed group under matrix multiplication [19].

| \( G_n \) |
|----------------|
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
| \( G_n \) |
Example 1. vector-based representations.

Over the generator set the terms a set of generators stabilizer matrix separately using a binary vector of size n. Therefore, an arbitrary stabilizer matrix is uniquely specified. The proof of Theorem 1 shows that they are transposition, which swaps two rows of the matrix, and multiplication, which left-multiplies one row with another. Such operations do not modify the stabilizer state and resemble the steps performed during a Gaussian elimination procedure. Several row-echelon (standard) forms for stabilizer generators along with relevant algorithms to obtain them have been introduced in the literature [3, 12, 19].

Canonical stabilizer matrices. Observe from Example 1 that, although stabilizer states are uniquely determined by their stabilizer group, the set of generators may be selected in different ways. Any stabilizer matrix can be rearranged by applying sequences of elementary row operations in order to obtain the row-reduced echelon form structure depicted in Figure 2b. This particular stabilizer-matrix structure defines a canonical representation for stabilizer states [10, 12]. The elementary row operations that can be performed on a stabilizer matrix are transposition, which swaps two rows of the matrix, and multiplication, which left-multiplies one row with another. Such operations do not modify the stabilizer state and resemble the steps performed during a Gaussian elimination procedure.

Stabilizer-circuit simulation. The computational-basis states are stabilizer states that can be represented by the stabilizer-matrix structure depicted in Figure 2. In this matrix form, the ± sign of each row along with its corresponding Zj-literal designates whether the state of the jth qubit is |0⟩ (+) or |1⟩ (−). Suppose we want to simulate circuit C. Stabilizer-based simulation first initializes M to specify some basis state. Then, to simulate the action of each gate U ∈ C, we conjugate each row Qi of M by U†. We require that the conjugation UQiU† maps to another string of Pauli literals so that the resulting matrix 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 [12, 19]. Table 2 lists mappings for the H, P and CNOT gates.

Example 2. Suppose we simulate a CNOT gate on

\[ |\psi⟩ = \sqrt{2} \left| 00 \right⟩ + \sqrt{2} |11⟩ \]

Then, to simulate the action of each gate U ∈ C, we conjugate each row Qi of M by U†. We require that the conjugation UQiU† maps to another string of Pauli literals so that the resulting matrix 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 [12, 19]. Table 2 lists mappings for the H, P and CNOT gates.

Example 2. Suppose we simulate a CNOT gate on |ψ⟩ = |00⟩ + |11⟩. Since stabilizer states are uniquely determined by their stabilizer group, the set of generators may be selected in different ways. Any stabilizer matrix can be rearranged by applying sequences of elementary row operations in order to obtain the row-reduced echelon form structure depicted in Figure 2b. This particular stabilizer-matrix structure defines a canonical representation for stabilizer states [10, 12]. The elementary row operations that can be performed on a stabilizer matrix are transposition, which swaps two rows of the matrix, and multiplication, which left-multiplies one row with another. Such operations do not modify the stabilizer state and resemble the steps performed during a Gaussian elimination procedure.

Several row-echelon (standard) forms for stabilizer generators along with relevant algorithms to obtain them have been introduced in the literature [3, 12, 19].

Stabilizer-circuit simulation. The computational-basis states are stabilizer states that can be represented by the stabilizer-matrix structure depicted in Figure 2. In this matrix form, the ± sign of each row along with its corresponding Zj-literal designates whether the state of the jth qubit is |0⟩ (+) or |1⟩ (−). Suppose we want to simulate circuit C. Stabilizer-based simulation first initializes M to specify some basis state. Then, to simulate the action of each gate U ∈ C, we conjugate each row Qi of M by U†. We require that the conjugation UQiU† maps to another string of Pauli literals so that the resulting matrix 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 [12, 19]. Table 2 lists mappings for the H, P and CNOT gates.

Example 2. Suppose we simulate a CNOT gate on |ψ⟩ = |00⟩ + |11⟩. Since stabilizer states are uniquely determined by their stabilizer group, the set of generators may be selected in different ways. Any stabilizer matrix can be rearranged by applying sequences of elementary row operations in order to obtain the row-reduced echelon form structure depicted in Figure 2b. This particular stabilizer-matrix structure defines a canonical representation for stabilizer states [10, 12]. The elementary row operations that can be performed on a stabilizer matrix are transposition, which swaps two rows of the matrix, and multiplication, which left-multiplies one row with another. Such operations do not modify the stabilizer state and resemble the steps performed during a Gaussian elimination procedure.

Several row-echelon (standard) forms for stabilizer generators along with relevant algorithms to obtain them have been introduced in the literature [3, 12, 19].

Stabilizer-circuit simulation. The computational-basis states are stabilizer states that can be represented by the stabilizer-matrix structure depicted in Figure 2. In this matrix form, the ± sign of each row along with its corresponding Zj-literal designates whether the state of the jth qubit is |0⟩ (+) or |1⟩ (−). Suppose we want to simulate circuit C. Stabilizer-based simulation first initializes M to specify some basis state. Then, to simulate the action of each gate U ∈ C, we conjugate each row Qi of M by U†. We require that the conjugation UQiU† maps to another string of Pauli literals so that the resulting matrix 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 [12, 19]. Table 2 lists mappings for the H, P and CNOT gates.
Using the stabilizer representation, $M_\psi = [\begin{pmatrix} +X \chi \\ +Z \zeta \end{pmatrix}] \to CNOT \to M'_\psi = [\begin{pmatrix} +X \chi' \\ +Z \zeta' \end{pmatrix}]$. The rows of $M'_\psi$ stabilize $|\psi\rangle$ if $(|0\rangle + |1\rangle)/\sqrt{2}$ as required.

Since H, P and CNOT gates are directly simulated via the stabilizer formalism, these gates are also known as stabilizer gates and any circuit composed exclusively of such gates is called a unitary stabilizer circuit. Table 2 shows that at most two columns of $M$ are updated when a Clifford (stabilizer) gate is simulated. Therefore, such gates are simulated in $\Theta(n)$ time. Furthermore, for any pair of Pauli operators $P$ and $Q$, $PQP^\dagger = (-1)^c Q$, where $c = 0$ if $P$ and $Q$ commute, and $c = 1$ otherwise. Thus, Pauli gates can also be simulated in linear time as they only permute the phase vector of the stabilizer matrix.

**Theorem 2:** An $n$-qubit stabilizer state $|\psi\rangle$ can be obtained by applying a stabilizer circuit to the $|0\rangle^\otimes n$ computational-basis state.

**Proof:** The work in [1] represents the generators using a tableau, and then shows how to construct a canonical stabilizer circuit $C$ from the tableau. We refer the reader to [1] Theorem 8 for details of the proof. Algorithms for obtaining more compact canonical circuits are discussed in [11].

**Corollary 3:** An $n$-qubit stabilizer state $|\psi\rangle$ can be transformed by Clifford gates into the $|00\ldots0\rangle$ computational-basis state.

**Proof:** Since every stabilizer gate can be produced by applying some unitary stabilizer circuit $C$ to the $|0\rangle^\otimes n$ state, it suffices to reverse $C$ to perform the inverse transformation. To reverse a stabilizer circuit, reverse the order of gates and replace every $P$ gate with $PPP$. The stabilizer formalism also admits measurements in the computational basis [12]. Conveniently, the formalism avoids the direct computation of measurement operators and inner products (Section 2). However, the updates to $M$ for such gates are not as efficient as for Clifford gates. 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. We can tell these cases apart in $\Theta(n)$ time by searching for $X$ or $Y$ literals in the $j$th column of $M$. 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 $p(1) = 1$).

**Example 3.** Suppose we have state $|1\rangle$, which is stabilized by $-Z$. Conjugating the stabilizer by the Phase gate yields $P(-Z)P^\dagger = -Z$. However, in the state-vector representation, $|1\rangle \mapsto i |1\rangle$. Thus the global phase $i$ is not maintained by the stabilizer.

Since global phases are unobservable, they do not need to be maintained when simulating a single stabilizer state. However, in Section 4 we show that such phases must be maintained when dealing with stabilizer-state superpositions, where global phases become relative.

### 3 Stabilizer Frames: Data Structure and Algorithms

The Clifford gates by themselves do not form a universal set for quantum computation [1], [19]. However, the Hadamard and Toffoli (TOF) gates do [2]. To simulate TOF and other non-Clifford gates, we extend the formalism to include the representation of arbitrary quantum
states as superpositions of stabilizer states.

**Example 4.** Recall from Section 2.2 that computational-basis states are stabilizer states. Thus, any one-qubit state $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$ is a superposition of the stabilizer states $|0\rangle$ and $|1\rangle$. In general, any state decomposition in a computational basis is a stabilizer superposition.

Suppose $|\psi\rangle$ in Example 4 is an unbiased state such that $\alpha_0 = i^k \alpha_1$, where $k = \{0, 1, 2, 3\}$. Then $|\psi\rangle$ can be 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. To this end, we leverage the following observation and derive a compact data structure for representing stabilizer-state superpositions.

**Observation 2.** Given an $n$-qubit stabilizer state $|\psi\rangle$, there exists an orthonormal basis including $|\psi\rangle$ and consisting entirely of stabilizer states. One such basis is obtained directly from the stabilizer of $|\psi\rangle$ by changing the signs of an arbitrary, non-empty subset of generators of $S(|\psi\rangle)$, i.e., by modifying the phase vector of the stabilizer frame for $|\psi\rangle$. Thus, one can produce $2^n - 1$ additional orthogonal stabilizer states. Such states, together with $|\psi\rangle$, form an orthonormal basis. This basis is unambiguously specified by a single stabilizer state, and any one basis state specifies the same basis.

**Definition 1.** An $n$-qubit stabilizer frame $F$ is a set of $k \leq 2^{n}$ stabilizer states $\{|\psi_j\rangle\}_{j=1}^k$ that forms an orthogonal subspace basis in the Hilbert space. We represent $F$ by a pair consisting of (i) a stabilizer matrix $M$ and (ii) a set of distinct phase vectors $\{\sigma_j\}_{j=1}^k$, where $\sigma_j \in \{-1, 1\}^n$. We use $M^{\sigma_j}$ to denote the ordered assignment of the elements in $\sigma_j$ as the $(-1)$-phases of the rows in $M$. Therefore, state $|\psi_j\rangle$ is represented by $M^{\sigma_j}$. The size of the frame, which we denote by $|F|$, is equal to $k$.

Each phase vector $\sigma_j$ can be viewed as a binary (0-1) encoding of the integer index that denotes the respective basis vector. Thus, when dealing with 64 qubits or less, a phase vector can be compactly represented by a 64-bit integer (modern CPUs also support 128-bit integers). To represent an arbitrary state $|\Psi\rangle$ using $F$, one additionally maintains a vector of complex amplitudes $a = (a_1, \ldots, a_k)$, which corresponds to the decomposition of $|\Psi\rangle$ in the basis $\{|\psi_j\rangle\}_{j=1}^k$ defined by $F$, i.e., $|\Psi\rangle = \sum_{j=1}^k a_j |\psi_j\rangle$ and $\sum_{j=1}^k |a_j|^2 = 1$. Observe that each $a_j$ forms a pair with phase vector $\sigma_j$ in $F$ since $|\psi_j\rangle \equiv M^{\sigma_j}$. Any stabilizer state can be viewed as a one-element frame.

**Example 5.** Let $|\Psi\rangle = a_1 (|00\rangle + |01\rangle) + a_2 (|10\rangle + |11\rangle)$. Then $|\Psi\rangle$ can be represented by the stabilizer frame $F$ depicted in Figure 3.

We now describe several frame operations that are useful for manipulating stabilizer-state superpositions.

**ROTATE ($F$, $U$).** Consider the stabilizer basis $\{|\psi_j\rangle\}_{j=1}^k$ defined by frame $F$. A stabilizer or Pauli gate $U$ acting on $F$ maps such a basis to $\{U |\psi_j\rangle = e^{i\theta_j} |\phi_j\rangle\}_{j=1}^k$, where $e^{i\theta_j}$ is the global phase of stabilizer state $|\phi_j\rangle$. Since we obtain a new stabilizer basis that spans the same subspace, this operation effectively rotates the stabilizer frame. Computationally, we perform a frame rotation as follows. First, update the stabilizer matrix associated with $F$ as per Section 2.2. Then, iterate over the phase vectors in $F$ and update each one accordingly (Table 2).

Let $a = (a_1, \ldots, a_k) \in \mathbb{C}^k$ be the decomposition of $|\Psi\rangle$ onto $F$. Frame rotation simulates the action of Clifford gate $U$ on $|\Psi\rangle$ since

$$U |\Psi\rangle = \sum_{j=1}^k a_j U |\psi_j\rangle = \sum_{j=1}^k a_j e^{i\theta_j} |\phi_j\rangle$$

Observe that the global phase $e^{i\theta_j}$ of each $|\phi_j\rangle$ becomes relative with respect to $U |\Psi\rangle$. Therefore, our approach requires that we compute such phases explicitly in order to maintain a consistent representation.

**Global phases of states in $F$.** Recall from Theorem 2 that any stabilizer state $|\psi\rangle = C |00\ldots0\rangle$ for some stabilizer circuit $C$. To compute the global phase of $|\psi\rangle$, one keeps track of the global factors generated when each gate in $C$ is simulated in sequence. In the context of frames, we maintain the global phase of each state in $F$ using the amplitude vector $a$. Let $M$ be the matrix associated with $F$ and let $\sigma_j$ be the $(-1)$-phase vector in $F$ that forms a pair with $a_j \in a$. When simulating Clifford gate $U$, each $a_j$ is updated as follows:

1. Set the leading phases of the rows in $M$ to $\sigma_j$.
2. Obtain a basis state $|b\rangle$ from $M$ and store its non-zero amplitude $\beta$. 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. Consider the matrix representation of $U$ and the vector representation of $\beta |b\rangle$, and compute $U (\beta |b\rangle) = \beta' |b'\rangle$ via matrix-vector multiplication.
4. Obtain $|b'\rangle$ from $UM^{\sigma_j}$ and store its amplitude $\gamma \neq 0$.
5. Compute the global factor generated as $a_j = (a_j \cdot \beta') / \gamma$.

By Observation 4, $M$ needs to be in row-echelon form (Figure 2) in order to sample the computational-

$$|\Psi\rangle = a_1 (|00\rangle + |01\rangle) + a_2 (|10\rangle + |11\rangle)$$

$$a = (a_1, a_2)$$

$$\begin{bmatrix} Z & I \\ I & X \end{bmatrix}$$

$$M \rightarrow M^{\sigma_1} = + \begin{bmatrix} Z & I \\ I & X \end{bmatrix} \equiv |00\rangle + |01\rangle$$

$$M \rightarrow M^{\sigma_2} = - \begin{bmatrix} Z & I \\ I & X \end{bmatrix} \equiv |10\rangle + |11\rangle$$

Fig. 3. A two-qubit stabilizer state $|\Psi\rangle$ whose frame representation uses two phase vectors.
basis amplitudes $|b\rangle$ and $|b'\rangle$. Thus, simulating gates with global-phase maintenance would take $O(n^3|F|)$ time for $n$-qubit stabilizer frames. To improve this, we introduce a simulation invariant.

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

Since Clifford gates act on 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 $n$-qubit stabilizer frame, the overall runtime for simulating a single Clifford gate is $O(n^2 + n|\mathcal{F}|)$ since one can memoize the updates to $\mathcal{M}$ required to compute each $a_j$.

Another advantage of maintaining this invariant is that the outcome of deterministic measurements (Section 2.2) can be decided in time linear in $n$ since it eliminates the need to perform Gaussian elimination.

**COFACTOR($\mathcal{F}$, $c$).** This operation facilitates measurement of stabilizer-state superpositions and simulation of non-Clifford gates using frames. (Recall from Section 2 that post-measurement states are also called cofactors.) Here, $c \in \{1, 2, \ldots, n\}$ is the cofactor index. Let $\{|\psi_j\rangle\}_{j=1}^k$ be the stabilizer basis defined by $\mathcal{F}$. Frame cofactoring maps such a basis to $\{|\psi_j^{\text{cof}}\rangle\}_{j=1}^k$. Therefore, after a frame is cofactored, its size either remains the same (qubit $c$ was in a deterministic state and thus one of its cofactors is empty) or doubles (qubit $c$ was in a superposition state and thus both cofactors are non-empty). We now describe the steps required to cofactor $\mathcal{F}$.

1) Check $\mathcal{M}$ associated with $\mathcal{F}$ to determine whether qubit $c$ is a random (deterministic) state. (Section 2.2)

2a) Suppose qubit $c$ is in a deterministic state. Since $\mathcal{M}$ is maintained in row-echelon form (Invariant 1) no frame updates are necessary.

2b) In the randomized-outcome case, apply the measurement algorithm from Section 2.2 to $\mathcal{M}$ while forcing the outcome to $x \in \{0, 1\}$. This is done twice – once for each $|x\rangle$-cofactor, and the row operations performed on $\mathcal{M}$ are memoized each time.

3) Let $\sigma_j$ be the $(\pm)$-phase vector in $\mathcal{F}$ that forms a pair with $a_j$. Iterate over each $(\sigma_j, a_j)$ pair and update its elements according to the memoized operations.

4) For each $(\sigma_j, a_j)$, insert a new phase vector-amplitude pair corresponding to the cofactor state added to the stabilizer basis.

Similar to frame rotation, the runtime of cofactoring is linear in the number of phase vectors and quadratic in the number of qubits. However, after this operation, the number of phase vectors (states) in $\mathcal{F}$ will have grown by a (worst case) factor of two. Furthermore, any state $|\Psi\rangle$ represented by $\mathcal{F}$ is invariant under frame cofactoring.

**Example 6.** Figure 4 shows how $|\Psi\rangle = |000\rangle + |010\rangle + |100\rangle + |110\rangle$ is cofactored with respect to its first two qubits. A total of four cofactor states are obtained.

### 4 Simulating Quantum Circuits with Stabilizer Frames

Let $\mathcal{F}$ be the stabilizer frame used to represent the $n$-qubit state $|\Psi\rangle$. Following our discussion in Section 3 any stabilizer or Pauli gate can be simulated directly via frame rotation.

Suppose we want to simulate the action of $TOF_{c_1, c_2, t}$, where $c_1$ and $c_2$ are the control qubits, and $t$ is the target. First, we decompose $|\Psi\rangle$ into all of its four cofactors (Section 2) over the control qubits to obtain the following equal superposition of orthogonal states:

$$|\Psi\rangle = \frac{1}{2}\left(\left|\Psi_{c_1 c_2 = 00}\right\rangle + \left|\Psi_{c_1 c_2 = 01}\right\rangle + \left|\Psi_{c_1 c_2 = 10}\right\rangle + \left|\Psi_{c_1 c_2 = 11}\right\rangle\right)$$

Here, we assume the most general case where all the $c_1 c_2$ cofactors are non-empty. The number of states in the superposition obtained could also be two (one control qubit has an empty cofactor) or one (both control qubits have empty cofactors). After cofactoring, we compute the action of the Toffoli as,

$$TOF_{c_1, c_2, t} |\Psi\rangle = \left(\left|\Psi_{c_1 c_2 = 00}\right\rangle + \left|\Psi_{c_1 c_2 = 01}\right\rangle + \left|\Psi_{c_1 c_2 = 10}\right\rangle + X_t \left|\Psi_{c_1 c_2 = 11}\right\rangle\right) / 2$$

where $X_t$ is the Pauli gate (NOT) acting on target $t$. We simulate Equation 2 with the following frame operations. (An example of the process is depicted in Figure 4).

1) COFACTOR($\mathcal{F}$, $c_1$).

2) COFACTOR($\mathcal{F}$, $c_2$).

3) Let $Z_j$ be the Pauli operator with a $Z$ literal in its $j^{th}$ position and $I$ everywhere else. Due to Steps 1 and 2, the matrix $\mathcal{M}$ associated with $\mathcal{F}$ must have two rows of the form $Z_{c_1}$ and $Z_{c_2}$. Let $u$ and $v$ be the indices of such rows, respectively. For each phase vector $\sigma_j\in\{1, \ldots, |\mathcal{F}|\}$, if the $u$ and $v$ elements of $\sigma_j$ are both -1 (i.e., if the phase vector corresponds to the $|\Psi_{c_1 c_2 = 11}\rangle$ cofactor), flip the value of element $t$ in $\sigma_j$ (apply $X_t$ to this cofactor).

Controlled-phase gates $R(\alpha)_{c_1, c_2}$ can also be simulated using stabilizer frames. This gate applies a phase-shift factor of $e^{i\alpha}$ if both the control qubit $c_1$ and target qubit $t$ are set. Thus, we compute the action of $R(\alpha)_{c_1, c_2}$ as,

$$R(\alpha)_{c_1, c_2} |\Psi\rangle = \left(\left|\Psi_{c_1 t = 00}\right\rangle + e^{i\alpha} \left|\Psi_{c_1 t = 10}\right\rangle\right) / 2$$

![Fig. 4. Simulation of $TOF_{c_1, c_2, t} |\Psi\rangle$ using a stabilizer-state superposition (Equation 2). Here, $c_1 = 1, c_2 = 2$ and $t = 3$. Amplitudes are omitted for clarity and the $\pm$ phases are prepended to matrix rows. The $X$ gate is applied to the third qubit of the $|\Psi_{c_1 c_2 = 11}\rangle$ cofactor.](image-url)
Equation \(3\) can be simulated via frame-based simulation using a similar approach as discussed for TOF gates. Let \(\{a_1, \ldots, a_{|\mathcal{F}|}\}\) be the decomposition of \(|\Psi⟩\) onto \(\mathcal{F}\). First, cofactor \(\mathcal{F}\) over the \(c\) and \(t\) qubits. Then, for any phase vector \(e_{j} \in \{1, \ldots, |\mathcal{F}|\}\) that corresponds to the \(|\Psi_{e_j=11}\rangle\) cofactor, set \(a_j = a_j e^{i\theta_j}\). 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.

**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 \(a_i\), and thus the overall measurement outcome may have a non-uniform distribution. Formally, let \(|\Psi⟩ = \sum_i a_i |\psi_i⟩\) be the superposition of states represented by \(\mathcal{F}\), the probability of observing outcome \(x \in \{0, 1\}\) upon measuring qubit \(m\) is,

\[
p(x)\Psi = \sum_{i=1}^{k} |a_i|^2 \langle \psi_i | P^m_x | \psi_i ⟩ = \sum_{i=1}^{k} |a_i|^2 p(x)\psi_i,
\]

where \(P^m_x\) denotes the measurement operator in the computational basis \(x\) as discussed in Section 2. The outcome probability for each stabilizer state \(p(x)\psi_i\) is computed as outlined in Section 2.2. Once we compute \(p(x)\psi_i\), we flip a (biased) coin to decide the outcome and cofactor the frame such that only the states that are consistent with the measurement remain in the frame.

Prior work on simulation of non-Clifford gates using the stabilizer formalism can be found in [1] where the authors represent a quantum state as a sum of \(O(4^{dk})\) density-matrix terms while simulating \(k\) non-Clifford operations acting on \(d\) distinct qubits. In contrast, the number of states in our technique is \(O(2^k)\) although we do not handle density matrices and perform more sophisticated bookkeeping.

**Technology-dependent gate decompositions.** Stabilizer-frame simulation can be specialized to quantum technologies that are represented by libraries of primitive gates (and, optionally, macro gates) with restrictions on qubit interactions as well as gate parallelism and scheduling. The work in [17] describes several primitive gate libraries for different quantum technologies including quantum dot, ion trap and superconducting systems. Such libraries can be incorporated into frame-based simulation by decomposing the primitive gates into linear combinations of Pauli or Clifford operators, subject to qubit-interaction constraints. For example, the 
\[ T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} \] gate and its inverse, which are primitive gates in most quantum machine descriptions, can be simulated as \(T_i|\Psi⟩ = (|\Psi⟩ + e^{i\pi/4}|\Psi⟩/\sqrt{2})\).

**Multiframe simulation.** Although a single frame is sufficient to represent a stabilizer-state superposition \(|\Psi⟩\), one can sometimes tame the exponential growth of states in \(|\Psi⟩\) by constructing a multiframe representation. Such a representation cuts down the total number of states required to represent \(|\Psi⟩\), thus improving the scalability of our technique. Our experiments in Section 6 show that, when simulating ripple-carry adders, the number of states in \(|\Psi⟩\) grows linearly when multiframes are used but exponentially when a single frame is used. To this end, we introduce an additional frame operation.

**COALESCE(\(\mathcal{F}\)).** 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 \(M\) of a frame in row-echelon form (Invariant 1), examining the phases corresponding to \(Z_k\)-rows (\(Z\)-literal in \(k^\text{th}\) column and \(I^\text{'s}\) in all other columns) allows us to identify the columns in \(M\) that need to be modified in order to coalesce candidate pairs.

More generally, suppose \(\langle \sigma_r, \sigma_j \rangle\) are a pair of phase vectors from the same \(n\)-qubit frame. Then \(\langle \sigma_r, \sigma_j \rangle\) is considered a candidate iff it has the following properties:

(i) \(\sigma_r\) and \(\sigma_j\) are equal up to \(m \leq n\) entries corresponding to \(Z_k\)-rows (where \(k\) is the qubit the row stabilizes), and
(ii) \(\sigma_r\) and \(\sigma_j\) are the frame amplitudes paired with \(\sigma_r\) and \(\sigma_j\) for some \(d \in \{0, 1, 2, 3\}\) (where \(\sigma_r\) and \(\sigma_j\) are the frame amplitudes paired with \(\sigma_r\) and \(\sigma_j\)). Let \(e = \{e_1, \ldots, e_m\}\) be the indices of a set of differing phase-vector elements, and let \(v = \{v_1, \ldots, v_n\}\) be the qubits stabilized by the \(Z_k\)-rows identified by \(e\). The steps in our coalescing procedure are:

1) Sort phase vectors such that candidate pairs with differing elements \(e\) are next to each other.
2) Coalesce candidates into a new set of phase vectors \(\sigma'\).
3) Create a new frame \(\mathcal{F}'\) consisting of \(\sigma'\) and matrix \(\mathcal{M}'\), where \(\mathcal{C} = \text{CNOT}_{e_1,v_1} \cdots \text{CNOT}_{e_d,v_d} \mathcal{M} \text{CNOT}_{e_1,v_1}' \cdots \text{CNOT}_{e_d,v_d}'\).
4) Repeat Steps 2–3 until no candidate pairs remain.

During simulation, we execute the coalescing operation after the set of phase vectors in a frame is expanded via cofactoring. Therefore, the choice of \(e\) (and thus \(\nu\)) is driven by the \(Z_k\)-rows produced after a frame is cofactored. (Recall that cofactoring modifies \(M\) such that each cofactored qubit \(k\) is stabilized by a \(±Z_k\) operator.)

The output of our coalescing operation is a list of \(n\)-qubit frames \(\mathcal{F} = \{\mathcal{F}_1', \mathcal{F}_2', \ldots, \mathcal{F}_r'\}\) (i.e., a multiframe) that together represent the same superposition as the original input frame \(\mathcal{F}\). The size of the multiframe generated is half the number of phase vectors in the input frame. The runtime of this procedure is dominated by Step 1. Each phase-vector comparison takes \(\Theta(\nu)\) time. Therefore, the runtime of Step 1 and our overall coalescing procedure is \(O(nk \log k)\) for a single frame with \(k\) phase vectors.

**Example 7.** Suppose we coalesce the frame \(\mathcal{F}\) depicted in Figure 5. Candidate pairs are \(\langle \sigma_1, \sigma_2 \rangle\) and \(\langle \sigma_3, \sigma_4 \rangle\), with \(e = \{2\}\) and \(e = \{2, 3\}\), respectively. To obtain \(\mathcal{F}_1\),...
extend our coalescing operation as follows:

\[ a = (5, 5, 5, 5), \]
\[ M = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \]
\[ F = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \]

![Diagram](image)

**Fig. 5.** Example of how a multiframe representation is derived from a single-frame representation.

which takes advantage of similarities across the (canonical) matrices in \( F \) to avoid expensive inner-product computations in many cases. We note that, when simulating quantum circuits that exhibit significant structure, \( F \) contains similar stabilizer matrices with equivalent rows (Pauli operators). Let \( M = \{ M_1, \ldots, M_k \} \) be the set of \( n \)-qubit stabilizer matrices in \( F \). Our heuristic keeps track of a set of Pauli operators \( P = \{ P_1, P_2, \ldots, P_{k\leq n} \} \), that form an intersection across the matrices in \( M \).

**Example 8.** Consider the multiframe from Figure 5. The intersection \( P \) consists of \( ZII \) (first row of both \( M \)).

By Observation 2 if two phase vectors (states) have different entries corresponding to the Pauli operators in \( P \), then the states are orthogonal and no inner-product computation is required. For certain practical instances, including the benchmarks described in Section 6, we obtain a non-empty \( P \) and our heuristic proves effective. When \( P \) is empty or the phase-vector pair is equivalent, we use the algorithm from [11] to verify orthogonality. Therefore, in the worst case, checking pairwise orthogonality of the states in \( F \) takes \( O(n^3 k^2) \) time for a multiframe that represents a \( k \)-state superposition.

The second task to consider when maintaining Invariant 2 is the orthogonalization of the states in \( F = \{ F_1, F_2, \ldots, F_k \} \) when our check fails. To accomplish this, we iteratively apply the **COFACTOR** operation to each frame in \( F \) in order to decompose \( F \) into a single frame. At each iteration, we select a *pivot qubit* \( p \) based on the composition of Pauli literals in the corresponding column. We apply the **COFACTOR** \(( \mathcal{F}, p \) \) operation only if there exists a pair of matrices in \( F \) that contain a different set of Pauli literals in the pivot column.

Observe that the order of pivot selection does not impact the performance of orthogonalization. Each iteration of the algorithm can potentially double the number of states in the superposition. Since the algorithm terminates when a single frame remains, the resulting states

**Orthogonality of multiframe.** We introduce the following invariant to facilitate simulation of quantum measurements on multiframe.

**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.

Given multiframe \( F = \{ F_1, \ldots, F_k \} \), one needs to consider two separate tasks in order to maintain Invariant 2. The first task is to verify the pairwise orthogonality of the states in \( F \). The orthogonality of two \( n \)-qubit stabilizer states can be checked using the inner-product algorithm described in [11], which takes \( O(n^3) \) time. To improve this, we derive a heuristic based on Observation 2.
are represented by distinct phase vectors and are therefore pairwise orthogonal. In the worst case, all \( n \) qubits are selected as pivots and the resulting frame constitutes a computational-basis decomposition of size \( 2^n \).

The overall simulation flow of our frame-based techniques is shown in Figure 6 and implemented in our software package Quipu.

**Example 9.** Figure 7 depicts the main steps of the Quipu simulation flow for a small non-Clifford circuit.

## 5 Parallel Simulation

Unlike other techniques based on compact representations of quantum states (e.g., using BDD data structures \[26\]), most frame-based operations are inherently parallel and lend themselves to multi-threaded implementation. The only step in Figure 6 that presents a bottleneck for a parallel implementation is the orthogonalization procedure, which requires communication across frames. All other processes at both the single- and multi-frame levels can be executed on large subsets of phase vectors independently.

We implemented a multithreaded version of Quipu using the C++11 thread support library. Each worker thread is launched via the `std::async()` function. Figure 8 shows our wrapper function for executing calls to `std::async()`. The `async_launch()` function takes as input: (i) a frame operation (`Func f`), (ii) a range of phase-vector elements defined by `Iter begin` and `Iter end`, and (iii) any additional parameters (`Params... p`) required for the frame operation. Furthermore, the function returns a vector of `std::future` – the C++11 mechanism for accessing the result of an asynchronous operation scheduled by the C++ runtime support system. The workload (number of phase vectors) of each thread is distributed evenly across the number of cores in the system (MTHREAD). The results from each thread are joined only when orthogonalization procedures are performed since they require communication between multiple threads. This is accomplished by calling the `std::future::get()` function on each future. All Clifford gates and measurements are simulated in parallel.

## 6 Empirical Validation

We tested a single-threaded and multi-threaded versions 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.

We used a straightforward implementation of the state-vector model using an array of complex amplitudes to perform functional verification of: (i) all benchmarks with < 30 qubits and (ii) Quipu output for such benchmarks. We simulated such circuits and checked for equivalence among the resultant states and operators \[25\]. In the case of stabilizer circuits, we used the equivalence-checking method described in \[1\].

**Stabilizer circuits.** We compared the runtime performance of single-threaded 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 \[15\]. For each \( n \), we generated the circuits as follows: fix a parameter \( \beta > 0 \); then choose \( \beta[n \log n] \) 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 \( \beta \) ranging from 0.6 to 1.2 in increments of 0.1. Figure 9 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. Therefore, the performance of both simulators 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. Nonetheless, Figure 9 shows that Quipu is asymptotically as fast as CHP when simulating stabilizer circuits that contain a linear number of measurements. The multithreaded speedup in Quipu for non-Clifford circuits is not readily available for stabilizer circuits.

**Ripple-carry adders.** Our second benchmark set consists of \( n \)-bit ripple-carry (Cuccaro) adder \[7\] circuits, which often appear as components in many arithmetic circuits \[18\]. The Cuccaro circuit for \( n = 3 \) is shown in Figure 10. 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 \( 2^n \) computational-basis states. Figure 11 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\[1\] specific to circuit simulation \[26\]. When \( n < 15 \), QPLite 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. Memory usage for both Quipu and QPLite was nearly unchanged for these benchmarks. Quipu consumed 4.7MB on average while QPLite consumed almost twice as much (8.5MB).

We ran the same benchmarks using single and multi-

frames. The number of states in the superposition grows exponentially in \( n \) for a single frame, but linearly in \( n \) when multiple frames are allowed. This is because TOF gates produce large equal superpositions that are effectively compressed by our coalescing technique. Since our frame-based algorithms require \( \text{poly}(k) \) time for \( k \) states in a superposition, Quipu simulates Cuccaro

frames.
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 while Quipu’s runtime grows quadratically (best quadratic fit \( f(x) = 0.5248x^2 - 15.815x + 123.86 \) with \( R^2 = .9986 \)).

The work in [18] 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 [11].

**Quantum Fourier transform (QFT) circuits.** Our third benchmark set consists of circuits that implement 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 \( x_1, 2, \ldots, m = \sum_{k=1}^{m} x_k 2^{-k} \). The action of the QFT on this input state can be expressed as:

\[
|x_1 \ldots x_n \rangle = \frac{1}{\sqrt{2^n}} \left( |0 \rangle + e^{2\pi i x_0} |1 \rangle \right) \otimes \left( |0 \rangle + e^{2\pi i x_1 \cdot x_0} |1 \rangle \right) \otimes \cdots \otimes \left( |0 \rangle + e^{2\pi i x_n \cdot x_{n-1} \cdots x_0} |1 \rangle \right) \tag{4}
\]

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 [12]. 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. Figure [13] shows average runtime and memory usage for both Quipu and QPLite on QFT instances for \( n = \{10, 12, \ldots, 20\} \). Quipu runs approximately \( 10 \times \) faster than QPLite on average and consumes about 96% less memory. For these benchmarks, we observed that the number of states in our multiframe runtimes was \( 2^n-1 \). This is because controlled-\( R(\alpha) \) gates produce biased superpositions (Section [4]) that cannot be effectively compressed using our coalescing procedure. Therefore, as Figure [13] shows, the runtime and memory requirements of both Quipu and QPLite grow exponentially in \( n \) for QFT instances. However, Quipu scales to 24-qubit instances while QPLite scales to only 18 qubits. The multi-threaded version of Quipu exhibited roughly a \( 2 \times \) speedup and used a comparable amount of memory on a four-core Xeon server.

We compared Quipu to a straightforward implementation of the state-vector model using an array of complex amplitudes. Such non-compact data structures can be streamlined to simulate most quantum gates (e.g., Hadamard, controlled-\( R(\alpha) \)) with limited runtime overhead, but scale to only around 30 qubits due to poor memory scaling. Our results showed that Quipu was approximately \( 3 \times \) slower than an array-based implementation when simulating QFT instances. However, such implementations cannot take advantage of circuit structure and, unlike Quipu and QPLite, do not scale to instances of stabilizer and arithmetic circuits with > 30 qubits (Figures [9] and [11]).

**Fault-tolerant (FT) circuits.** Our next benchmark set consists of circuits that, in addition to preparing encoded quantum states, implement procedures for performing FT quantum operations [9], [19], [23]. 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 Clifford gate transversally across QECC-registers as shown in Figure [14]. Non-Clifford gates need to be implemented using a FT architecture that often requires ancilla qubits, measurements and correction procedures conditioned on measurement outcomes. Figure [15] shows a circuit that implements a FT-Toffoli operation [23]. Each line represents a 5-qubit register based on the DiVincenzo/Shor [13] code, and each gate is applied transversally. The state \( \ket{cat} = (\ket{0^5} + \ket{1^5})/\sqrt{2} \) 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 \( H, CNOT, H \), with control \( i \) and target \( j \). \( Z \) gates are implemented as \( P^2 \).

We implemented FT benchmarks for the half-adder and full-adder circuits (Figure [16]) as well as for computing \( f(x) = b^x \mod 15 \). Each circuit from Figure [17] implements \( f(x) \) with a particular coprime base value.

Fig. 12. The three-qubit QFT circuit.
TABLE 3
Average time and memory needed by Quipu and QPLite to simulate several quantum FT circuits. The second column shows the QECC used to encode \( k \) logical qubits into \( n \) physical qubits. Benchmarks with (*) use the 5-qubit DiVincenzo/Shor code [9] instead of the 3-qubit bit-flip code. We used the \(|00\ldots0\rangle\) input state for all benchmarks. The top numbers from each row correspond to direct simulation of Toffoli gates (Section 4) and the bottom numbers correspond to simulation via the decomposition from Figure 18. Shaded rows are Clifford circuits for mod-exp.

| FAULT-TOLERANT CIRCUIT | QECC \([n, k]\) | NUM. QUBITS | NUM. GATES | RUNTIME (SECS) | MEMORY (MB) | MAX SIZE (\(\psi\)) |
|------------------------|-------------|-------------|-------------|---------------|--------------|-------------------|
| toffoli*               | \([15, 3]\) | 45          | 125         | 72.45         | 137.02       | 2816              |
| halfadd*               | \([15, 3]\) | 45          | 125         | 72.45         | 137.02       | 2816              |
| fulladd*               | \([20, 4]\) | 80          | 320         | 84.96         | 139.44       | 16384             |
| 2<sup>2</sup>mod15     | \([18, 6]\) | 81          | 396         | 4.8 hrs       | 118.17       | 22528             |
| 4<sup>2</sup>mod15*    | \([30, 6]\) | 30          | 30          | <0.01         | 6.14         | 12.01             |
| 7<sup>2</sup>mod15     | \([18, 6]\) | 81          | 396         | 11.25 hrs     | 12.41        | 13.29             |
| 8<sup>2</sup>mod15     | \([18, 6]\) | 81          | 396         | >24 hrs       | 134.05       | 14.77             |
| 11<sup>2</sup>mod15*   | \([30, 6]\) | 30          | 25          | <0.01         | 6.14         | 12.01             |
| 13<sup>2</sup>mod15    | \([18, 6]\) | 81          | 396         | 11.28 hrs     | 11.85        | 12.25             |
| 14<sup>2</sup>mod15*   | \([30, 6]\) | 30          | 40          | <0.01         | 6.14         | 12.01             |

Fig. 14. Transversal implementation of a stabilizer circuit acting on three-qubit QECC registers.

Fig. 15. Fault-tolerant implementation of a Toffoli gate.

Fig. 16. Adder circuits implemented in our benchmarks.

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 \(|1, 2, 4, 8\rangle\) or \(|1, 4, 1, 4\rangle\).

Technology-dependent circuits. Section 4 outlines how Quipu supports primitive gate libraries, especially for quantum-optical systems where Clifford gates are considered primitive [17]. Therefore, to simulate FT circuits for photonic systems, it suffices to decompose \( TOFF \) gates into sequences of Hadamard, CNOT and \( T \) gates as shown in Figure 15. Table 3 reports simulations of our FT benchmarks using such decompositions. Since the total number of gates is larger, Quipu is roughly more robust 5-qubit code in our larger benchmarks. Our results in Table 3 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. For FT benchmarks that consist of stabilizer circuits (shaded rows), the QuIDD representation remains compact and utilizes half as much memory as our frame representation.

Table 3 also shows that our coalescing technique is very effective as the maximum size of the stabilizer-state superposition is much smaller when multiple frames are used. Since the total number of states observed is relatively small, the multithreaded version of Quipu exhibited similar runtime and memory requirements for these benchmarks.
new Clifford miter – linear combinations of Clifford operators that represent a specific quantum circuit. Clifford miter can speed up formal verification by exploiting similarities in circuits and the fast equivalence-checking algorithms from [1], [11].

Section 4 described how Quipu incorporates quantum machine descriptions in the form of primitive gate libraries to simulate technology-dependent circuits. This method can benefit from new decompositions for library gates into linear combinations of Pauli or Clifford operators. Such decompositions can be obtained on the fly when simulating original gates one-by-one in sequence. They can also be precomputed and used for a compiled version of the original circuit, where scheduling can be optimized for parallelism and architecture constraints.

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

7 Conclusions

In this work, we developed new techniques for quantum-circuit simulation based on superpositions of stabilizer states, avoiding shortcomings in prior work [1]. To represent such superpositions compactly, we designed a new data structure called a stabilizer frame. We implemented stabilizer frames and relevant 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, but simulates certain quantum arithmetic circuits in polynomial time and space for input states consisting of equal superpositions of computational-basis states. In contrast, QuIDDPro takes exponential time on such instances. We simulated 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. In particular, our multithreaded implementation of Quipu exhibited a 2× speed up on a four-core server.

Future Directions. The work in [27] describes an equivalence-checking method for quantum circuits based on the notion of a reversible miter – a counterpart of miter circuits used in equivalence-checking of digital circuits. An attractive direction for future work is deriving

4× slower as compared to direct simulation of TOFF gates. QPLite takes > 24 hours to simulate several of these benchmarks while Quipu takes only several seconds since the majority of the gates introduced by the decomposition from Figure 18 are Clifford gates.

Fig. 17. Mod-exp with $M = 15$ implemented as (2, 4)-LUTs [18] for several coprime base values. Negative controls are shown with hollow circles.

Fig. 18. The Toffoli gate and its decomposition into one-qubit and CNOT gates [19], Figure 4.9.

Fig. 18. The Toffoli gate and its decomposition into one-qubit and CNOT gates [19], Figure 4.9.

References

[1] S. Aaronson, D. Gottesman, “Improved Simulation of Stabilizer Circuits,” Phys. Rev. A, vol. 70, no. 5, 052328 (2004).
[2] D. Aharonov. “A Simple Proof that Toffoli and Hadamard are Quantum Universal,” arXiv:0301040 (2003).
[3] K. M. R. Audenaert, M. B. Plenio, “Entanglement on Mixed Stabilizer States: Normal Forms and Reduction Procedures,” New J. Phys., vol. 7, no. 170 (2005).
[4] A. Barenco et al., “Approximate Quantum Fourier Transform and Decoherence,” Phys. Rev. A, vol. 54, pp. 139–146 (1996).
[5] O. Boncalo et al., “Using Simulated Fault Injection for Fault Tolerance Assessment of Quantum Circuits,” Proc. Sim. Symp., pp. 213–220 (2007).
[6] D. Coppersmith, “An Approximate Fourier Transform Useful in Quantum Factoring,” arXiv:0201067 (2002).
[7] S. A. Cuccaro et al., “A New Quantum Ripple-carry Addition Circuit,” arXiv:0410184v1 (2004).
[8] K. De Raedt et al., “Massively Parallel Quantum Computer Simulator,” Comp. Phys. Comm., vol. 176, no. 2 (2007).
[9] D. P. DiVincenzo, P. W. Shor, “Fault-Tolerant Error Correction with Efficient Quantum Codes”, Phys. Rev. Lett., vol. 77, no. 3260 (1996).
[10] I. Djordjevic, Quantum Information Processing and Quantum Error Correction: an Engineering Approach, Academic press (2012).
[11] H. J. Garcia, I. L. Markov, A. W. Cross, “On the Geometry of Stabilizer States,” Quant. Info. and Comp., vol. 14, no. 7–8 (2014).
[12] D. Gottesman, “The Heisenberg Representation of Quantum Computers,” arXiv:9807006v1 (1998).
[13] L. Grover, “A Fast Quantum Mechanical Algorithm for Database Search,” Symp. on Theory of Comp., pp. 212–219 (1996).
[14] T. W. Hungerford, Algebra, Springer (1974).
[15] N. Isailovic et al., “Interconnection Networks for Scalable Quantum Computers,” Inter. Symp. on Comp. Arch., pp. 366–377 (2006).
[16] E. Knill et al., “Randomized Benchmarking of Quantum Gates,” Phys. Rev. A., vol. 77, no. 1 (2007).
[17] C. Lin, A. Chakrabarti, N.K. Jha, “Optimized Quantum Gate Library for Various Physical Machine Descriptions,” IEEE Trans. on VLSI Sys., vol. 21, no. 11 (2013).
[18] I. L. Markov, M. Saeedi, “Constant-optimized Quantum circuits for Modular Multiplication and Exponentiation,” Quant. Info. and Comp., vol. 12, no. 5 (2012).
[19] M. A. Nielsen, I. L. Chuang, Quantum Computation and Quantum Information, Cambridge University Press (2000).
[20] K. M. Obenland, A. M. Despain, “A Parallel Quantum Computer Simulator,” arXiv:9804039 (1998).
[21] B. Oemer, http://tph.tuwien.ac.at/~oemer/qcl.html (2003).
[22] M. Oskin, F. T. Chong, I L. Chuang, “A Practical Architecture for Reliable Quantum Computers,” IEEE Computer, vol. 35, no. 1 (2002).
[23] J. Preskill, “Fault Tolerant Quantum Computation,” Introduction to Quantum Computation, World Scientific (1998). arXiv:9712048.
[24] P. Shor, “Polynomial-time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer,” SIAM J. Comput, vol. 26, no. 5 (1997).
[25] K. M. Svore, A. V. Aho, A. W. Cross, I. L. Chuang, I. L. Markov, “A Layered Software Architecture for Quantum Computing Design Tools,” IEEE Computer, vol. 39, no. 1 (2006).
[26] G. F. Viamontes, I. L. Markov, J. P. Hayes, Quantum Circuit Simulation, Springer (2009).
[27] S. Yamashita, I. L. Markov, “Fast Equivalence-checking for Quantum Circuits” Quant. Info. and Comp., vol. 9, no. 9–10 (2010).