Fast Equivalence-checking for Quantum Circuits

SHIGERU YAMASHITA
Ritsumeikan University, 1-1-1 Noji Higashi
Kusatsu, Shiga 525-8577, Japan
ger@cs.ritsumei.ac.jp

IGOR L. MARKOV
University of Michigan, 2260 Hayward St.
Ann Arbor, Michigan 48109, U.S.A.
imarkov@eecs.umich.edu

March 9, 2010

Abstract— We perform formal verification of quantum circuits by integrating several techniques specialized to particular classes of circuits. Our verification methodology is based on the new notion of a *reversible miter* that allows one to leverage existing techniques for simplification of quantum circuits. For *reversible circuits* which arise as runtime bottlenecks of key quantum algorithms, we develop several verification techniques and empirically compare them. We also combine existing quantum verification tools with the use of SAT-solvers. Experiments with circuits for Shor's number-factoring algorithm, containing thousands of gates, show improvements in efficiency by four orders of magnitude.

1 Introduction

Quantum circuits are considered among promising emerging technologies beyond conventional semiconductors [22], and significant progress was made recently in practical implementations. In August 2009 "*researchers at the [US] National Institute of Standards and Technology ... demonstrated continuous quantum operations using a trapped-ion processor*" [6] that maintained quantum bits in hyperfine states of beryllium ions for up to 15 seconds at a time. An implementation account of NIST's quantum processor [4] shows that the design of even two-qubit circuits relies on software tools, similar in spirit to logic-synthesis and optimization tools used today to design digital logic circuits. Software tools for quantum circuit design are the primary subject of our work. We note that a number of largely unrelated technologies for quantum circuits are currently pursued by researchers. In addition to ion traps, several photonic realizations of Shor's number-factoring algorithm have been reported since 2007 [7, 8], including a single-chip circuit [15]. This motivates our focus on technology-independent tools for quantum circuits, which can be useful for many specific technologies and admit further adaptations.

Quantum circuits often operate on quantum states that contain exponentially large superpositions, making quantum simulation, as well as circuit design and analysis on conventional computers very challenging. To this end, a layered software architecture for quantum computing design tools was outlined in [18]. Our work focuses on one

such task — verifying the results of quantum circuit transforms, e.g., adaptations of technology-independent quantum circuits to linear device architectures, such as ion traps [2, 9]. Given a circuit that is known to be correct, one seeks to prove that a new circuit optimized for a given physical technology is equivalent to the original circuit.

Past research in equivalence-checking for quantum circuits developed computational techniques based on Binary Decision Diagrams (BDDs) [12, 20, 21]. These techniques can represent some exponentially large complex-valued vectors and matrices using compact graphs. Quantum operations are then modeled by graph algorithms whose complexity scales with graph size rather than with the size of superpositions or the amount of entanglement present. However, these algorithms are much slower than those for equivalence-checking of conventional digital logic and do not scale to useful instances of Shor's algorithm.

An important observation is that a typical quantum algorithm consists of heterogeneous modules [14] that favor different computational techniques for equivalence-checking. This motivates the development of a new verification methodology that invokes the most appropriate technique for each module type and assembles the results. Our methodology relies on a new concept, introduced in Sec. 3 and called *a reversible miter* — a natural counterpart of *miter circuits* used in equivalence-checking of digital electronic circuits. In conjunction with existing techniques for iterative circuit simplification [5, 10, 16], reversible miters can drastically reduce the size and complexity of circuits under verification, especially when such circuits bear some structural resemblance (e.g., when adapting textbook circuits to specific quantum-computing architectures).

In Sec. 4 we develop an high-performance equivalence-checking for quantum circuits. Our method is *adaptive* in the sense that it utilizes multiple techniques appropriate for different classes of quantum circuit modules. In this context, we study *reversible circuits* which are a subset of quantum circuits that map conventional 0-1 bit-strings into other such bit-strings. In particular, the largest module in Shor's number-factoring algorithm [17] — *modular exponentiation* — is implemented as a reversible circuit [11] (acting on entangled quantum states), exceeds all other modules asymptotically in size, and thus requires most attention of CAD tools. To verify such logic modules, we adapt conventional state-of-the-art techniques [13, 23] in several ways, and significantly scale up quantum equivalence checking. Empirical comparisons in Sec. 4.1 confirm that properties of reversible circuits can enable much faster SAT-based equivalence-checking. However, conventional techniques cannot be applied to, e.g., the *Quantum Fourier Transform (QFT)*. Therefore, we also study equivalence-checking of circuits with non-conventional gates (we call these circuits *properly-quantum*), and the integration of heterogeneous techniques.

Our contributions can be summarized as follows.

- *Reversible miters* for equivalence-checking of quantum circuits, and their integration with circuit simplification.
- The use of SAT-based equivalence checking and its integration with BDD-based techniques.
- Adaptive equivalence-checking for quantum circuits that integrates reversible miters, circuit simplification, as well as SAT- and BDD-based techniques.

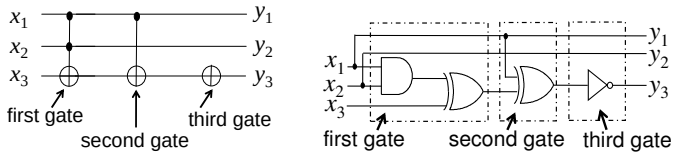

Figure 1: A reversible circuit and its irreversible realization. Dashed boxes show how reversible gates are represented by one-output gates used in digital logic.

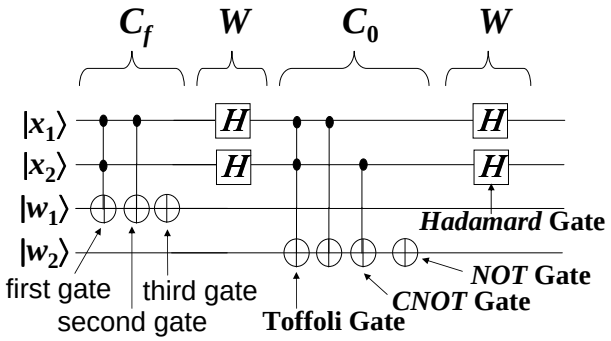

Figure 2: A properly-quantum circuit — one iteration of Grover algorithm. Circuit modules labeled C_f, W, C_0 and W are composed by concatenation.

2 Notation and Preliminaries

Recall that, when acting on conventional bits, gates NOT, CNOT and TOFFOLI can be implemented using NOT, XOR and AND gates as shown in Fig. 1. In the quantum case, they exchange basis states, which is why their matrices contain only 0s and 1s. As these gates obey the same algebraic rules in both cases, we term them *conventional gates*. In comparison, the matrix of the Hadamard gate contains $1/\sqrt{2}$, and its functionality cannot be expressed in Boolean logic. Therefore we call such gates **properly-quantum**. *Each properly-quantum gate maps at least one 0-1 input combination (basis state) to a quantum superposition of more than one basis state.* Circuits that include properly-quantum gates are also called properly-quantum. Properly quantum circuits are necessary to generate entanglement, perform quantum error correction and achieve computational speed-up over traditional algorithms. An example is given in Fig. 2. As we show below, many reversible circuits without properly-quantum gates can be verified relatively easily in practice using a state-of-the-art equivalence-checking tools for conventional logic circuits based on solving instances of Boolean SATisfiability. Modern SAT-solvers exploit structure in application-derived instances, and modern equivalence-checkers automatically identify and exploit similarities in the circuits whose equivalence is checked.

Many quantum algorithms contain large, application-specific sections dedicated to the computation of Boolean functions. In order to embed conventional computation into the quantum domain, it must be made reversible, and standard procedures exist for

such transformations [14]. The resulting circuits do not create entanglement, but can be applied to superposition superposition states. Leveraging this quantum parallelism in useful applications is difficult, but can be illustrated by Shor's polynomial-time algorithm for number-factoring [14, 17]. This algorithm is dominated by a reversible module that performs modular exponentiation [11] before the Quantum Fourier Transform (QFT). We call such circuits without properly-quantum gates specifically **reversible circuits** in this paper. A gate library used for reversible circuits is *universal* iff it can express any (conventional) reversible transformation by combining multiple copies of gates involved. The most common such gate library consists of NOT, CNOT, and Toffoli gates. Since the algebraic properties of the gates in reversible circuits do not involve quantum phenomena, we can calculate the logic functions realized at each point in a circuit, as is normally done in conventional logic synthesis and verification. For example, we can calculate the function at wire x_3 of the circuit shown in Fig. 1 after the third gate as $y_3 = x_3 \oplus x_1x_2 \oplus x_1 \oplus 1$.

3 Reversible Miters

To check the equivalence of two combinatorial digital logic circuits, C_1 and C_2, one checks if the conventional *miter circuit* [13] shown in the left-hand side of Fig. 3 implements the constant-0 function. In other words, every pair of outputs are XOR'ed, all XOR-outputs are OR'ed together, and the resulting Circuit-SAT instance is converted to CNF-SAT using known techniques (a number of optimized reductions have been proposed recently with large circuits in mind). Conventional miters can be constructed for reversible circuits by treating them as AND/OR/NOT circuits, except that such miters will not be reversible. Therefore, we introduce *reversible miters* which can handle reversible and properly-quantum circuits equally well, and can benefit from simplification of reversible circuits [5, 10, 16].

3.1 Properties of Quantum Circuits

In the following, the $\cdot$ symbol represents a concatenation of two circuits (or gates) as illustrated in Fig. 2. Observe that for quantum or reversible circuits C_1 and C_2, the concatenated circuit $C_1 \cdot C_2$ is of the same kind. Such circuits can also be structurally reversed.

Observation 1 *Given a quantum (or reversible) circuit $C = g_1 \cdot g_2 \cdot \cdots \cdot g_k$ where g_i is a gate, its copy where all gates are inverted and put in the reverse order, i.e., $g_k^{-1} \cdot \cdots \cdot g_2^{-1} \cdot g_1^{-1}$, implements the inverse transformation to what C implements. We therefore denote it by C^{-1}.*

For example, for a circuit C shown in the left-hand side of Fig. 1, the circuit $C \cdot C^{-1}$ is given in the right-hand side of Fig. 3. Note that NOT, CNOT, and Toffoli gates are their own inverses (which explains their choice as library gates). The circuit $C \cdot C^{-1}$ is equivalent to an *empty circuit*. This can be confirmed by iteratively cancelling out pairs of mutually-inverse adjacent gates. Namely, in the right-hand side of Fig. 3, the third and the fourth gates can be removed at once. Then, the second and the fifth gates,

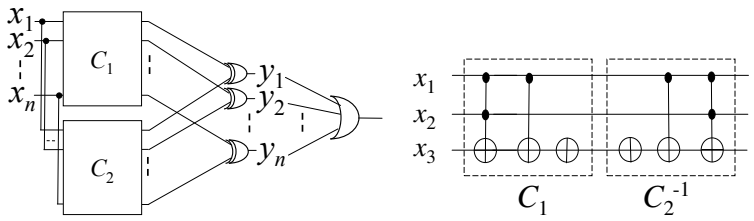

Figure 3: Miter circuits: conventional and reversible.

followed by the first and the last gates. This observation motivates our new notion of *reversible miters*.

3.2 Reversible Miter Circuits

Definition 1 *Given two quantum (or reversible) circuits C_1 and C_2, their* reversible miter *is defined to be one of the following circuits: $C_1 \cdot {C_2}^{-1}$, ${C_2}^{-1} \cdot C_1$, $C_2 \cdot {C_1}^{-1}$, ${C_1}^{-1} \cdot C_2$.*

In particular, for conventional miters one needs to check that the output functions implement the constant 0 function, whereas for reversible miters one checks that each output bit is equivalent to a corresponding input bit. Namely, C_1 and C_2 are functionally equivalent if and only if all of their reversible miters implement the identity transformation. In particular, if one miter implements the identity, then so do the remaining miters. If $C_1 = C_2$, then straightforward circuit simplification [5, 10, 16] cancels out all gates, resulting in an empty circuit. Some of the variant miters enable more cancellations than others, e.g., if C_1 and C_2 differ only in their first segments, $C_2 \cdot C_1^{-1}$ exhibits many gate cancellations.

Reversible miters speed up equivalence-checking by exploiting similarities in circuits by two distinct mechanisms.

3.2.1 Local Simplification of Reversible Miters

When two conventional circuits end with identical gate sequences, one cannot cancel out these sequences because of observability don't-cares introduced by them. However, reversible circuits do not experience don't-cares, and identical suffixes always cancel out. Note that a reversible miter $C_1 \cdot C_2^{-1}$ places the last gate of C_1 next to the last gate of C_2. If these two gates cancel out, the second-to-last gates from C_1 and C_2 become adjacent, etc. Thus, no search is required to identify these gate cancellations, and they can be performed one at a time. Even if the last two gates are different, it may be possible to cancel out second-to-last gates, as long as the last and second-to-last gates do not act on the same (qu)bit lines. These are special cases of much more general *local simplifications* discussed in [5, 10, 16]. If C_1 and C_2 are identical, an empty circuit will result, but this outcome is also possible when local simplifications can prove equivalence of two structurally different circuits. A systematic procedure for applying

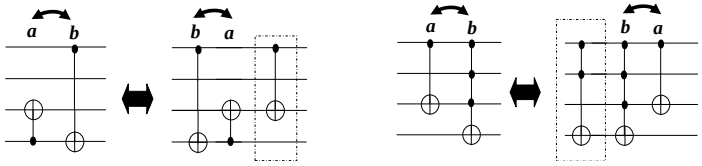

Figure 4: Gate swap (complicated case).

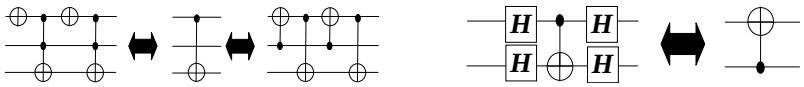

Figure 5: Equivalent circuit templates.

simplifications was introduced in [5]. Local simplifications in reversible circuits are particularly easy to perform, are fast and do not consume much memory [10,16]. In our experiments, even the simplest simplification rules can dramatically simplify reversible miters. More sophisticated simplifications from [5, 10, 16] provide an additional boost.

We experimented with the following simplification procedure. In a miter circuit, consider one gate at a time, search for a matching inverse, and try to move them together to facilitate cancellation. Any two gates can be swapped if they do not act on the same (qu)bit lines. Two adjacent NOT, CNOT or Toffoli gates can be swapped if the control bit of one gate is not the target bit of the other gate (same for properly-quantum controlled-U gates). A more sophisticated swapping rule (for NOT, CNOT, and Toffoli gates) is illustrated in Fig. 4.

In our procedure, for the purposes of equivalence-checking, we temporarily consider the miter circuit to be "circular" by connecting its outputs to its inputs. Namely, we allow moving the first gate to the end of the circuit, as illustrated in Fig. 6. This transformation does not change the equivalence of the entire circuit to the identity. In other words, if $g_1 \cdot g_2 \cdot \cdots \cdot g_{k-1} \cdot g_k = I$ (Identity), then $g_1^{-1} \cdot g_1 \cdot g_2 \cdot \cdots \cdot g_{k-1} \cdot g_k \cdot g_1 = g_1^{-1} \cdot I \cdot g_1 = g_1^{-1} \cdot g_1 = I$. Therefore, to check equivalence between $g_1 \cdot g_2 \cdot \cdots \cdot g_{k-1} \cdot g_k$ and I is the same as to check equivalence between $g_2 \cdot \cdots \cdot g_k \cdot g_1$ and I.

A variety of circuit-equivalence templates can be used with the above simplification procedure [10, 16, 19] to shrink the miter circuit. Such templates are known for both reversible and properly-quantum gates as shown in Fig. 5. For example, the transformation illustrated in Fig. 6 enables further simplification through the equivalence in Fig. 5 on the right.

3.2.2 Simplification of Canonical Forms

Iterative circuit simplification is not guaranteed to reduce $C_1 \cdot C_2^{-1}$ to the empty circuit in polynomial time when such a simplification is possible. Finding a short simplification may be time-consuming. Yet, when constructing canonical forms (ROBDDs or QuIDDs) of reversible miters, a different kind of simplification may occur. Sup-

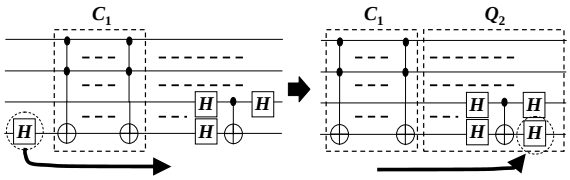

Figure 6: Transforming a miter circuit after simplification. Dashed boxes outline circuit modules, while horizontal dashed lines represent omitted qubits and gates.

pose that C_1 and C_2 end with functionally-equivalent but structurally distinct suffixes that do not admit local simplifications — an example is given in [16]. In other words $C_1 = A_1 \cdot B_1$ and $C_2 = A_2 \cdot B_2$ where $B_1 \approx B_2$. Then $C_1 \cdot C_2^{-1} = A_1 \cdot B_1 \cdot B_2^{-1} \cdot A_2^{-1} \approx A_1 \cdot A_2^{-1}$. As we traverse the miter $C_1 \cdot C_2^{-1}$, adding one gate at a time to the decision diagram (DD), the size of the intermediate DDs depends only on the transformation implemented by the current circuit prefix, i.e., the functions of the intermediate wires. The intermediate DD for $A_1 \cdot B_1 \cdot B_2^{-1}$ can be smaller than that for $A_1 \cdot B_1$ if $A_1 \cdot B_1 \cdot B_2^{-1} \approx A_1$. This phenomenon was observed in our experiments.

4 Equivalence-checking for Quantum Circuits

We now introduce equivalence-checking of quantum circuits based on several techniques appropriate for different classes of quantum circuits. The first class contains reversible circuits that arise as key modules in quantum algorithms.

4.1 Equivalence-checking for Reversible Circuits

To check the equivalence of two reversible circuits, C_1 and C_2, one can pursue two strategies. The first strategy is to check that the conventional miter implements the constant 0 function. A conventional miter can also be applied to reversible circuits as explained below. The second strategy is to represent the transformations performed by C_1 and C_2 in a canonical form which supports efficient equivalence-checking.

The latter strategy may use binary-decision diagrams (BDDs), such as ROBDDs, and QuIDDs [20] or QMDDs [12]. The former can be implemented with either decision diagrams or Boolean Satisfiability solvers by reducing Circuit-SAT to CNF-SAT. In particular, for conventional miters one needs to check that the output functions implement the constant 0 function. In addition to the basic SAT or BDD-based approaches, finding equivalent signals in two circuits is often very helpful [13]. Such techniques appear useful for reversible circuits as well, as shown in our experiments.

4.1.1 Using existing computational engines

ROBDD. Calculate the output functions of miter circuits, using ROBDD as the primary data structure. This technique cannot handle properly quantum circuits.

QuIDD. Build functional representations of given circuits C_1 and C_2, and check if the

results are identical. In particular, QuIDDPro [20, 21] builds multi-terminal decision diagrams called QuIDDs that can capture properly-quantum circuits.

SAT. Given two reversible circuits, construct a CNF-SAT formula that is satisfied only by those input combinations for which the two circuits produce different outputs. Then use a contemporary SAT solver [24] to check satisfiability.* We construct a CNF formula as follows. First we add a set of clauses for each gate in the miter circuit. The clauses should be satisfied only with the variable assignments that are consistent with the reversible gate. The readers familiar with SAT-based equivalence-checking can think of a CNOT gate as an XOR gate with a bypass wire, and of a Toffoli gate as an XOR, AND and a bypass. More efficient clause generation is illustrated below for a Toffoli gate whose control bits are x_1 and x_2, and target bit is x_3. Since the Toffoli gate does not modify two of its inputs, there is no need for separate output variables. We introduce only one new variable y_1 for the target bit. Then logical consistency is given by the condition $y_1 = (x_1 \cdot x_2) \oplus x_3$ which can be expressed by the following six clauses.

- Case $x_1 = 0$ or $x_2 = 0$. Clauses: $(x_1 + \overline{x_3} + y_1) \cdot (x_1 + x_3 + \overline{y_1}) \cdot (x_2 + \overline{x_3} + y_1) \cdot (x_2 + x_3 + \overline{y_1})$.
- Case $x_1 = x_2 = 1$. Clauses: $(\overline{x_1} + \overline{x_2} + x_3 + y_1) \cdot (\overline{x_1} + \overline{x_2} + \overline{x_3} + \overline{y_1})$.

In the next step, we add a set of clauses that are satisfied only by those variable combinations where some circuit output differs from the respective circuit input.

Here we can reuse some of the y variables introduced earlier. Let such a new variable corresponding to the i-th primary output be y_{O_i}. (If there is no target bit on the i-th bit-line, we do not introduce a new variable for the i-th primary output, i.e., it is obvious that the input and the output functions on the i-th bit-line are the same, and thus we do not add the following clauses.) We introduce a new variable z_i to express the functional consistency of the i bit-line. Namely, we consider that z_i becomes 1 only when $x_i \neq y_{O_i}$. For this condition, we add the following clauses.

- Case $z_i = 0$. Clauses: $(z_i + x_i + \overline{y_{O_i}}) \cdot (z_i + \overline{x_i} + y_{O_i})$.
- Case $z_i = 1$. Clauses: $(\overline{z_i} + x_i + y_{O_i}) \cdot (\overline{z_i} + \overline{x_i} + \overline{y_{O_i}})$.

Finally we add $(z_1 + z_2 + \cdots + z_n)$ where n is the number of bit-lines of the circuits. Since $z_i = 1$ mens that the input and the output functions on the i-th bit-line are different, the two circuits are different when $(z_1 + z_2 + \cdots + z_n)$ is satisfied. Therefore, the above construction generates a SAT formula that is satisfied only by those input combinations for which the corresponding outputs of two circuits produce different values. A CNF-SAT formula constructed for a miter grows linearly with the size of the miter. A key advantage of reversible miters is that they can be significantly smaller, due to gate cancellations and other circuit simplifications.

*Recall that NP-completeness relates to worst-case complexity and does not prevent fast solution of many application-derived SAT instances. In industrial applications, modern SAT solvers can often resolve CNF-SAT instances with hundreds of thousands variables in several hours, although small hard instances are also known.

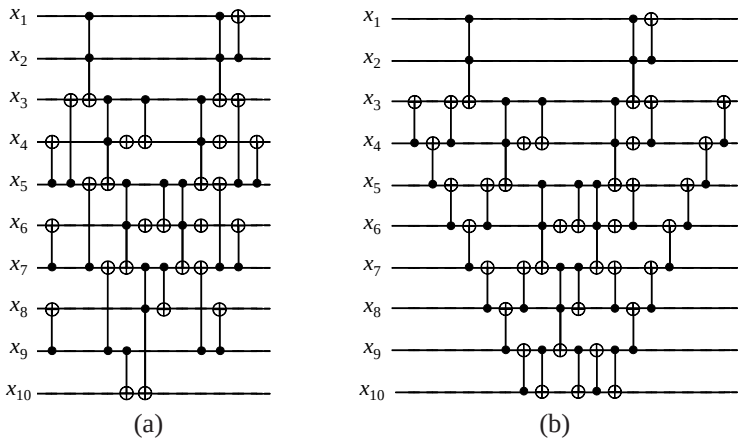

Figure 7: A ripple-carry adder circuit for n = 4 (a), and its LNN version (b).

4.1.2 State-of-the-art Combinational Equivalence Checking

SAT-based techniques can be dramatically improved through synergies with randomized functional simulation and through identifying intermediate equivalences. By hashing the results of random simulation, one finds candidate equivalent wires. If w_1 and w_2 are *not* equivalent, the counterexample returned by SAT is used to refine the results of functional simulation and often distinguishes other seemingly-equivalent pairs of wires. Once intermediate wires w_1 and w_2 are proven equivalent, all downstream gates are reconnected to w_1, and w_2 can be excluded from the SAT instance (along with some of its upstream gates). If potentially equivalent wires are selected in a topological order from the inputs, the impact of multiple circuit restructuring steps accumulates, until all output wire are proven equivalent or until an input combination is found that disproves the equivalence of outputs.

The state-of-the-art implementation of these techniques found in the Berkeley ABC system [23] (the "cec" command) features incremental SAT-solving and *fraiging* — a fast circuit-simplification technique based on hashing [13]. To use ABC, we construct a conventional (irreversible) circuit from a reversible circuit as shown in Fig. 1.

The impact of random-simulation techniques on SAT-based equivalence-checking can be illustrated by the example of multiplier circuits, which are known to confound both BDD-based and SAT-based computations. The case of equivalent multipliers is particularly difficult because it cannot be quickly concluded by finding (perhaps, by luck) input combinations that disprove the equivalence. However, if the two given multipliers are structurally similar and include many equivalent wires, then global equivalence can be proven quickly through a series of lemmata. Empirical data in Table 2 shows that on a 6-bit multiplier CEC (SAT-based combinatorial equivalence-checking) outperforms by far BDD-based and SAT-only methods.

Common benchmarks for reversible circuit synthesis can be verified in milliseconds by the above techniques. Therefore, we focus on scalable blocks of standard quantum

algorithms, whose optimization and equivalence-checking are critical to the success of quantum computers being designed today. More concretely, we performed experiments with n-bit *linear-nearest-neighbor (LNN)* CNOT gate circuits, a reversible ripple-carry adder circuit proposed in [1], *mesh* circuits [2] and reversible multipliers. Given a (qu)bit ordering, a linear-nearest-neighbor (LNN) CNOT gate circuit is a circuit which realizes the functionality of a CNOT gate with target and control bits k bits apart, by using only LNN gates (gates that operate only on adjacent qubits). As an example of the circuits in our study, we show a ripple-carry adder circuit for $n = 4$, and its LNN version in Fig. 7. Studies of LNN architectures are important because several promising implementations of quantum computation require the LNN architecture (also called the *spin-chain* architecture in the physics literature) and allow only adjacent qubits to interact directly. Thus, standard quantum circuits must be adapted to such architectures and modified to use only LNN gates. Specific transformations and LNN circuits have been developed [2, 9]. The overhead of the LNN architecture in terms of the number of gates is often limited by a small factor (3-5). Such physical-synthesis optimization motivates the need for equivalence-checking against the original, non-LNN versions. Using important components of Shor's algorithm [2, 14] — adders, meshes and multipliers — we build three types of equivalence-checking instances.

Same. Two equivalent circuits.

Different 1. Add ten random Toffoli gates at the end.

Different 2. Add ten random Toffoli gates at the beginning.

Our empirical data for CNOT, adder and mesh circuits exhibits essentially the same trends. Hence we report results only for adders in Table 1. Runtimes are reported in $seconds$ on a Linux system with a 2.40GHz Intel® Xeon™ CPU with 1GB RAM.

We implemented n-bit reversible multipliers using $5n$ bit-lines, including $2n$ bits for two inputs, $2n$ bits for the results, and n ancillae. E.g., the line $n = 6$ in the tables deals with 30-bit circuits. The n-bit adder circuit proposed in [1] uses $2n + 2$ qubits. Thus, the third column in Tables 1 and 2 shows the number of qubits in each circuit. The forth column shows the number of gates in each circuit. All methods other than "cec" timed out for $n = 8$, requiring more than 1,000s.

4.2 Checking Properly-Quantum Circuits

In this section we show that our proposed techniques can handle properly-quantum gates, but remain compatible with fast special-case methods.

4.2.1 Utility of Reversible Miters

Earlier sections focused on equivalence-checking of reversible circuits which appear in modules of quantum algorithms and require physical synthesis optimizations [2] that must be verified. However, other important modules in quantum algorithms, such as the *Quantum Fourier Transform (QFT)*, are properly-quantum, and conventional circuits, such as *modular exponentiation*, can be optimized for performance using properly-quantum gates. Fortunately, the utility of miters relies on (symbolic) cancellations of gates, and equally applies to reversible and properly-quantum circuits, unlike previously known techniques for verifying conventional digital circuits. Reduced properly-

Table 1: Adder verification performed by several techniques.

Case	n	♯qubits	♯gates	SAT	QuIDD	BDD	cec
Same	32	66	280	0.65	20.10	**0.03**	**0.19**
	64	130	568	2.91	115.85	**0.11**	**0.23**
	128	258	1144	11.71	771.20	**0.52**	**0.31**
Diff. 1	32	66	290	1.00	31.93	**0.04**	**0.02**
	64	130	578	5.16	212.57	**0.25**	**0.26**
	128	258	1154	15.25	$>$ 1,000	1.67	**0.38**
Diff. 2	32	66	290	1.09	40.40	0.09	**0.02**
	64	130	578	10.98	318.62	0.76	**0.03**
	128	258	1154	22.72	$>$ 1,000	9.88	**0.03**

Table 2: Multiplier verification performed by several techniques.

	n	♯qubits	♯gates	SAT	QuIDD	BDD	cec
Same	4	20	166	1.86	50.45	0.09	**0.00**
	6	30	411	392.74	$>$ 1,000	39.19	**0.01**
Diff. 1	4	20	176	**0.02**	72.84	**0.01**	**0.01**
	6	30	421	0.11	$>$ 1,000	**0.03**	**0.02**
Diff. 2	4	20	176	**0.02**	95.94	**0.01**	**0.02**
	6	30	421	0.17	$>$ 1,000	**0.01**	**0.02**

quantum miters can be verified using symbolic simulation with QuIDDPro [20] or QMDD software [12]. Using reversible miters as pre-processors can greatly decrease overall runtime. We empirically compare the following two methods.

With Local Simplification. Before invoking QuIDDPro, reduce the miter using local simplification.

W/o Local Simplification. Apply QuIDDPro directly to the miter.

For properly-quantum circuit benchmarks, we used QFT and *modular exponentiation* modules from circuits that implement Shor's factorization algorithm on an LNN architecture [2]. For each benchmark circuit with n inputs, we studied five cases (new gates were added in the middle).

Same. Two identical copies of a benchmark circuit.

Different 1. A circuit and its copy with one gate added.

Different 2. A circuit and its copy with two gates added.

Different 3. A circuit and its copy with one gate deleted.

Different 4. A circuit and its copy with two gates deleted.

In Tables 3 to 6 we report runtimes in seconds for local simplification of reversible miters and subsequent QuIDDPro calls, subject to a $1000s$ time-out. In the "Same" case, simplification alone proved equivalence. However, in the "Diff. 2" case, many gates remained after simplification and QuIDD runtimes were substantial. In all cases, local simplification improved overall runtimes.

For a more convincing example, we check equivalence between an LNN and non-

Table 3: Verifying QFT circuits without local simplification. Compare to Table 4.

n	Same		Diff. 1		Diff. 2		Diff. 3		Diff. 4	
	simp.	QuIDD	simp.	QuIDD	simp.	QuIDD	simp.	QuIDD	simp.	QuIDD
4	-	**0.15**	-	**0.15**	-	0.16	-	0.14	-	0.14
8	-	**1.75**	-	**1.80**	-	1.97	-	1.74	-	1.83
16	-	**> 1,000**	-	**> 1,000**	-	> 1,000	-	> 1,000	-	> 1,000
32	-	**> 1,000**	-	**> 1,000**	-	> 1,000	-	> 1,000	-	> 1,000
64	-	**> 1,000**	-	**> 1,000**	-	> 1,000	-	> 1,000	-	> 1,000

Table 4: Verifying QFT circuits with local simplification. Compare to Table 3.

n	Same		Diff. 1		Diff. 2		Diff. 3		Diff. 4	
	simp.	QuIDD	simp.	QuIDD	simp.	QuIDD	simp.	QuIDD	simp.	QuIDD
4	**0**	-	**0**	**0.03**	0	0.05	0	0.04	0	0.05
8	**0**	-	**0.01**	**0.03**	0	0.17	0	0.04	0	0.26
16	**0.05**	-	**0.07**	**0.05**	0.08	0.26	0.06	0.04	0.07	0.05
32	**0.73**	-	**1.11**	**0.04**	1.13	9.17	0.99	0.04	1.22	0.08
64	**17.29**	-	**24.32**	**0.05**	25.48	0.52	24.33	0.06	30.35	0.12
128	**354.52**	-	**366.2**	**0.04**	497.21	> 1,000	522.57	0.04	580.11	0.39

Table 5: Verifying modular multiplication w/o local simplification. Compare to Table 6.

n	Same		Diff. 1		Diff. 2		Diff. 3		Diff. 4	
	simp.	QuIDD	simp.	QuIDD	simp.	QuIDD	simp.	QuIDD	simp.	QuIDD
4	-	**> 1,000**	-	**> 1,000**	-	> 1,000	-	> 1,000	-	> 1,000
8	-	**> 1,000**	-	**> 1,000**	-	> 1,000	-	> 1,000	-	> 1,000

Table 6: Verifying modular multiplication with local simplification. Compare to Table 5

.

n	Same		Diff. 1		Diff. 2		Diff. 3		Diff. 4	
	simp.	QuIDD	simp.	QuIDD	simp.	QuIDD	simp.	QuIDD	simp.	QuIDD
4	**0.58**	-	**0.98**	**0.04**	1.07	0.85	0.98	0.05	1.02	0.39
8	**2.13**	-	**3.72**	**0.04**	3.69	0.37	3.73	0.04	3.45	1.19
16	**6.03**	-	**10.11**	**0.05**	11.29	> 1,000	11.16	0.05	11.26	5.73
32	**16.33**	-	**27.65**	**0.04**	27.49	3.68	27.21	0.05	27.83	0.04
64	**36.28**	-	**58.32**	**0.02**	59.27	0.56	60.91	0.05	60.13	1.33
128	**74.77**	-	**119.71**	**0.04**	120.98	1.88	120.83	0.05	121.59	52.55

LNN implementation (without *measurement gates*) of Shor's algorithm for factoring the number 15. These equivalent properly-quantum circuits include 2,732 gates for the non-LNN version and 5,120 gates for the LNN version. Their structure is very different. For equivalence-checking, we used QuIDDPro with and without local simplification, and these runs completed in 59.07s and 64095.22s, resp. The results confirm the effectiveness of local simplifications with reversible properly-quantum miters.

4.2.2 Boosting Verification by Using SAT-based Combinational Tools

Local simplification may leave many gates around, after which QuIDDPro tends to consume significant time and memory. However, if very few properly-quantum gates remain, a more lightweight verification procedure may be used. Generic symbolic simulators, such as QuIDDPro, do not scale (empirically) as well as leading-edge SAT-based combinational equivalence-checking (CEC) used in the Electronics industry to verify modern digital circuits (Sec. 4.1). Hence we leverage SAT-based tools to boost equivalence-checking of quantum circuits.

FOR TWO CIRCUITS C_1 AND C_2, WE DO THE FOLLOWING.

Step 1. Construct the miter circuit $C = C_1 \cdot C_2^{-1}$.
Step 2. Perform simplification of the miter circuit.
Step 3. If properly-quantum gates remain, go to Step 4, else invoke state-of-the-art SAT-based combinational equivalence-checking (the "cec" command of ABC system [23]) to tell if the miter circuit is equivalent to Identity.
Step 4. Find the longest sequence of conventional logic gates (NOT, CNOT, Toffoli) in the miter circuit. Label this sequence C_a. Let the simplified miter circuit be $Q_a \cdot C_a \cdot Q_b$.
Step 5. Transform $Q_a \cdot C_a \cdot Q_b$ to $C_a \cdot Q_b \cdot Q_a$. Note that $Q_a \cdot C_a \cdot Q_b = I$ (Identity) iff $C_a \cdot Q_b \cdot Q_a = I$ as shown in Sec. 3.2.1. Move conventional gates in $Q_b \cdot Q_a$ to the front of the miter as much as possible, creating a transformed miter $C'_a \cdot Q'_b$, where C'_a and Q'_b are a reversible circuit and a properly-quantum circuit, respectively.
Step 6. Check the functionality of Q'_b by lightweight iterated simulation. If it is not properly quantum, conclude that the miter circuit is not Identity. Else, go to Step 7.
Step 7. Exploit the functionality of Q'_b, and let C_b be a conventional circuit which corresponds to the exploited logic functionality. Then, check whether $C'_a \cdot C_b$ is Identity or not.

Suppose we have few properly-quantum gates as shown in the left-hand side of Fig. 6 where C_1 is relatively large. Then after Step 5, we can get the right-hand side circuit from the left-hand side circuit in Fig. 6. Our miter becomes $C_1 \cdot Q_2$ where C_1 is reversible but Q_2 is properly-quantum. This avoids a heavy-duty generic quantum simulator for C_1.

A key observation is that the functionality of Q'_b (at Step 6) should be classical (inverse of C'_a) if the entire miter is Identity. Thus, if Q'_b is properly-quantum, the miter circuit is not Identity. When Q'_b has few gates, this can be checked efficiently by a quantum generic simulator. By Step 7, properly-quantum gates are reduced, and we can use state-of-the-art SAT-based combinational equivalence-checking. By avoiding heavy-duty generic quantum simulation, our adaptive method can achieve significant speed-ups when C'_a is large.

To validate our method, we studied circuits implementing one iteration of Grover's quantum algorithm for search [3] as shown in Fig. 2. A particular step of the algorithm, called *the oracle*, is implemented with a reversible circuit module C_f based on a user-defined Boolean function f (search predicate). To make verification more challenging, we configured a search predicate that contains a multiplier circuit. We then created an equivalent variant of C_f by applying a global, rather than local, circuit transform. Namely, we applied a certain wire permutation on inputs of C_f and its inverse on outputs of C_0. This permutation was implemented by applying SWAP gates to (all) pairs of adjacent wires and then breaking down each SWAP gate into three CNOT gates, as described in Section 2. In our case study, the proposed procedure goes as follows.

Step 1. Construct the miter circuit $C = C_1 \cdot C_2^{-1} = C_f^1 \cdot W^1 \cdot C_0^1 \cdot W^1 \cdot (W^2)^{-1} \cdot (C_0^2)^{-1} \cdot (W^2)^{-1} \cdot (C_f^2)^{-1}$.

Step 2. Simplify the miter circuit. Because of the inserted SWAP gates (if we use only naive cancellation rules), we cannot cancel the two pairs of C_f^1 and $(C_f^2)^{-1}$, or C_0^1 and $(C_0^2)^{-1}$. But we can remove the sequence $W^1 \cdot (W^2)^{-1}$, reducing the miter to $C_f^1 \cdot W^1 \cdot C_0^1 \cdot (C_0^2)^{-1} \cdot (W^2)^{-1} \cdot (C_f^2)^{-1}$.

Step 3. Since properly-quantum gates remain, go to Step 4.

Steps 4 and 5. Move $(C_f^2)^{-1}$ to the input side of the circuit to maximize the conventional logic part in the prefix. The miter becomes $C'_a \cdot Q'_b$ where $C'_a = (C_f^2)^{-1} \cdot C_f^1$ and $Q'_b = W^1 \cdot C_0^1 \cdot (C_0^2)^{-1} \cdot (W^2)^{-1}$.

Steps 6. and 7. Using techniques described earlier, combine a quantum generic simulator (QuIDDPro [20, 21]) and state-of-the-art SAT-based combinational equivalence-checking (the "cec" command of ABC system [23]).

The above technique is compared to constructing a miter circuit and applying the symbolic simulator QuIDDPro [20, 21] to the miter. QuIDDPro alone does not finish in ten hours, but our technique completes in under seven seconds.

5 Conclusion and Future Work

We have studied several techniques for equivalence-checking of reversible circuits, including the new concept of reversible miters. In particular, we have observed that state-of-the-art SAT-based combinational equivalence-checking (cec) can be adapted to this context and outperforms generic quantum techniques. Basic BDD-based techniques usually outperform SAT-based techniques, but not cec. As is the case with ATPG, reversibility can significantly simplify equivalence-checking, while these simplifications are compatible with other techniques and amplify them. We then proposed an adaptive method to verify quantum circuits more efficiently than the existing quantum circuit verification tools by combining them with the state-of-the-art SAT-based combinational equivalence-checking tool for the conventional circuits. Experiments suggest that reversible miters are useful for the verification of reversible circuits as well as properly-quantum circuits.

References

[1] S. A. Cuccaro et al. A new quantum ripple-carry addition circuit. http://xxx.lanl.gov/abs/quant-ph/0410184, Los Alamos e-print, 2004.

[2] A. G. Fowler, S. J. Devitt and L. C. L. Hollenberg. Implementation of Shor's algorithm on a linear nearest neighbour qubit array. *Quantum Information and Computation*, 4(4):237–251, July 2004.

[3] L. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Physical Review Letters*, 79(2):325–328, July 1997.

[4] D. Hanneke et al. J. P. Home, J. D. Jost, J. M. Amini, D. Leibfried, D. J. Wineland Realisation of a programmable two-qubit quantum processor. Quantum Physics pre-print ARXIV.ORG:0908.3031, *National Inst. of Standards and Technology*, August 2009.

[5] K. Iwama, Y. Kambayashi and S. Yamashita. Transformation Rules for Designing CNOT-based Quantum Circuits. *DAC*, pp. 419–424, 2002.

[6] R. C. Johnson. NIST scales up quantum computing. *EE Times*, 6 Aug. 2009.

[7] B. P. Lanyon, T. J. Weinhold, N. K. Langford, M. Barbieri, D. F. V. James, A. Gilchrist and A. G. White. Experimental Demonstration of a Compiled Version of Shor's Algorithm with Quantum Entanglement. *Physical Review Letters*, 99, 250505, Dec. 2007.

[8] C-Y Lu, D. E. Browne, T. Yang and J-W Pan. Demonstration of a Compiled Version of Shor's Quantum Factoring Algorithm Using Photonic Qubits. *Physical Review Letters*, 99, 250504, Dec. 2007.

[9] D. Maslov. Linear depth stabilizer and quantum Fourier transformation circuits with no auxiliary qubits in finite-neighbor quantum architectures. *Physical Review A* 76, 052310, Nov. 2007.

[10] D. Maslov, G. W. Dueck, D. M. Miller and C. Negrevergne. Quantum circuit simplification and level compaction. *IEEE Trans. on CAD*, 27(3):436–444, Mar. 2008.

[11] R. Van Meter, K. M. Itoh, "Fast quantum modular exponentiation," *Physical Review A* 71(052320), May 2005.

[12] D. M. Miller and M. A. Thornton. QMDD: A decision diagram structure for reversible and quantum circuits. *IEEE Int'l Symp. on Multiple-Valued Logic*, p.30, May 2006.

[13] A. Mishchenko, S. Chatterjee, R. Brayton, N. Een. Improvements to combinational equivalence checking. *ICCAD*, pp. 836–843, 2006.

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

[15] A. Politi, J. C. F. Matthews and J. L. O'Brien. Shor's Quantum Factoring Algorithm on a Photonic Chip. *Science 4*, 325(5945):1221, Sept. 2009.

[16] A. K. Prasad, V. V. Shende, K. N. Patel, I. L. Markov and J. P. Hayes. Algorithms and data structures for simplifying reversible circuits. *ACM J. of Emerging Technologies in Computing*, 2(4):277–293, Oct. 2006.

[17] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[18] 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* 39(1): 74-83, 2006.

[19] R. R. Tucci. QC Paulinesia. http://xxx.lanl.gov/abs/quant-ph/0407215, Los Alamos e-print, 2004.

[20] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Improving gate-level simulation of quantum circuits. *Quantum Information Processing*, 2(5):347–380, 2003.

[21] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Equivalence checking of quantum circuits and states. *ICCAD*, pp. 69–74, 2007.

[22] J. Yoshida. 35 people, places and things that will shape the future. *EE Times*, February 29, 2008.

[23] The ABC Home Page. `http://www.eecs.berkeley.edu/~alanmi/abc/`.

[24] The MiniSat Home Page. `http://minisat.se/MiniSat.html`.