Assertion-based optimization of Quantum programs

Assertion-Based Optimization of Quantum Programs

THOMAS HÄNER[*], ETH Zürich, Switzerland
TORSTEN HOEFLER, ETH Zürich, Switzerland
MATTHIAS TROYER, Microsoft, USA

Quantum computers promise to perform certain computations exponentially faster than any classical device. Precise control over their physical implementation and proper shielding from unwanted interactions with the environment become more difficult as the space/time volume of the computation grows. Code optimization is thus crucial in order to reduce resource requirements to the greatest extent possible. Besides manual optimization, previous work has adapted classical methods such as constant-folding and common subexpression elimination to the quantum domain. However, such classically-inspired methods fail to exploit certain optimization opportunities across subroutine boundaries, limiting the effectiveness of software reuse. To address this insufficiency, we introduce an optimization methodology which employs annotations that describe how subsystems are entangled in order to exploit these optimization opportunities. We formalize our approach, prove its correctness, and present benchmarks: Without any prior manual optimization, our methodology is able to reduce, e.g., the qubit requirements of a 64-bit floating-point subroutine by 34×.

CCS Concepts: • **Hardware** → **Quantum computation**; • **Software and its engineering** → *Compilers*.

Additional Key Words and Phrases: quantum computing, quantum circuit optimization

ACM Reference Format:
Thomas Häner, Torsten Hoefler, and Matthias Troyer. 2020. Assertion-Based Optimization of Quantum Programs. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 133 (November 2020), 20 pages. https://doi.org/10.1145/3428201

1 INTRODUCTION

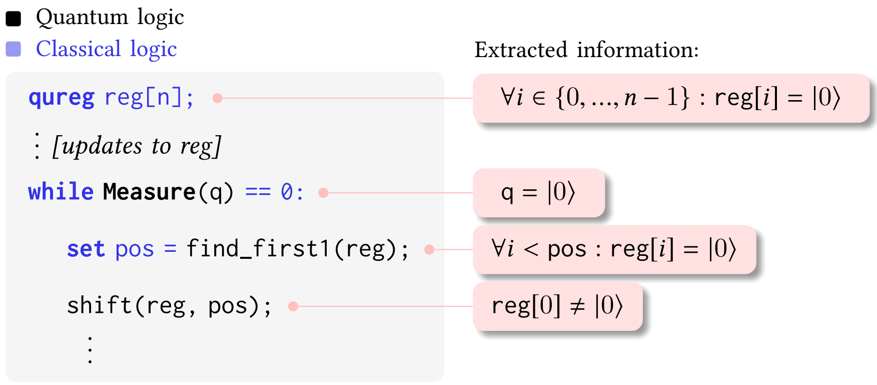
Quantum computers promise to solve certain computational tasks exponentially faster than classical computers. As a result, significant resources are being spent in order to make quantum computing become reality. In anticipation of the first quantum computers, the most promising applications are being identified and manually optimized for specific problems of practical interest, resulting in great resource savings [Gidney and Ekerå 2019; Häner et al. 2017; Hastings et al. 2015; Kutin 2006; Reiher et al. 2017]. Such improvements are crucial, given the considerable overhead due to quantum error correction [Fowler et al. 2012] and the difficulty of engineering large-scale quantum computers.

To identify promising applications for quantum computing, it is necessary to develop a detailed understanding of all components of the quantum algorithm being studied. One possible approach is to implement the algorithm in a quantum programming language, as this also enables testing and debugging. To this end, a host of software packages, programming languages, methodologies, and compilers for quantum computing have been developed [Chong et al. 2017; Green et al. 2013;

[*] Also with Microsoft Quantum, Zürich.

Authors' addresses: Thomas Häner, ETH Zürich, Switzerland; Torsten Hoefler, ETH Zürich, Switzerland; Matthias Troyer, Microsoft, USA.

2475-1421/2020/11-ART133
https://doi.org/10.1145/3428201



```
qureg reg[n];
⋮ [updates to reg]
while Measure(q) == 0:
    set pos = find_first1(reg);
    shift(reg, pos);
      ⋮
```

■ Quantum logic
■ Classical logic
Extracted information:
$\forall i \in \{0, \ldots, n-1\} : \mathrm{reg}[i] = |0\rangle$
$q = |0\rangle$
$\forall i < \mathrm{pos} : \mathrm{reg}[i] = |0\rangle$
$\mathrm{reg}[0] \neq |0\rangle$

Fig. 1. Example depicting a quantum program and the corresponding information that our methodology infers from the postconditions of both classical and quantum subroutines. It uses this information to exploit optimization opportunities that arise due to subroutine composition. Here, the `shift` subroutine may be optimized, as discussed in Section 5.

IBM 2018; JavadiAbhari et al. 2014; Paykin et al. 2017; Smith et al. 2016; Steiger et al. 2018; Svore et al. 2018]. In addition to providing the necessary layers of abstraction to facilitate software development, these packages include optimizing compilers that aim to reduce width and depth of the resulting quantum circuit, e.g., by merging operations at various layers of abstraction [Häner et al. 2018b]. Further optimization opportunities can be created by employing a set of commutation relations [Nam et al. 2018] to reorder operations. Moreover, several methods have been developed for exact circuit synthesis with certain optimality guarantees [Amy et al. 2013; Große et al. 2007, 2009; Kliuchnikov et al. 2013; Meuli et al. 2018]. While these methods alone are not suitable for optimization of large-scale quantum circuits, they can be combined with heuristics [Amy et al. 2014; Nam et al. 2018].

Despite these efforts, most of the progress made in, e.g., quantum chemistry has been due to manual optimization [Hastings et al. 2015; Jones et al. 2012]. This suggests that the capabilities of optimizing compilers may still be significantly improved; especially at higher levels of abstraction where manual optimizations are carried out today. Such optimization opportunities usually arise when two or more existing subroutines are combined instead of directly implementing the desired functionality more efficiently. Exploiting these opportunities is only possible when optimizing across the boundaries of both quantum and classical subroutines. To enable such transformations, an optimizing compiler requires access to additional information such as the circumstances under which a given subroutine is invoked. While compilers may not be able to infer the semantics of a given program and its subroutines, such information may be extracted from assertions, and then used for program *optimization*. Specifically, we propose to use the pre- and postconditions of each subroutine in order to gather information about the state of the quantum computer before and after completion of the subroutine. Fig. 1 depicts a mixed quantum/classical code example and annotations of postconditions and derived facts for both classical and quantum subroutines. The gathered information may then be combined, e.g., with conditions specifying the circumstances under which a given subroutine acts trivially. If these conditions are met, our methodology removes such operations and is thus able to reduce both space and time requirements of a given quantum program.

Contributions. We develop a formalism that allows us to express (1) how subsystems (subsets of qubits) of the quantum computer are entangled and (2) under what circumstances the program may be optimized. Our methodology then uses this information for quantum program optimization. We prove its correctness and present a prototype implementation that successfully reduces the quantum resource requirements of common arithmetic subroutines by up to 410× (for a 64-bit floating-point subroutine). The chosen subroutines are frequently used in a wide range of quantum algorithms, including Shor's algorithm for factoring [Shor 1994], HHL for solving linear systems of equations [Harrow et al. 2009], algorithms for quantum chemistry [Babbush et al. 2016], and Grover's algorithm [Grover 1996] when used for optimization. Our proposed automatic program-level optimizations are thus beneficial for a wide range of quantum computing applications.

Related work. By carrying out optimizations across different subroutines of the quantum program, our methodology complements available tools for lower-level circuit optimization and synthesis [Amy et al. 2014, 2013; Nam et al. 2018]. Crucially, our methodology enables more effective use of abstractions when implementing libraries for quantum computing because it is able to remove the resulting overheads. Furthermore, previous work on high-level quantum program optimization [Häner et al. 2018b; Steiger et al. 2018], which adapted common subexpression elimination and constant-folding to the quantum domain, cannot handle the examples we present in Section 5. Moreover, the verification of quantum programs has been addressed through the introduction of a quantum Hoare logic [Ying 2012]. In contrast, our methodology uses the information that it gathers from pre- and postconditions of subroutines for optimization purposes.

2 PRELIMINARIES

In this section, we provide some background on quantum computing and quantum programs. For a more in-depth treatment of quantum computing, we refer to the textbook by Nielsen and Chuang [Nielsen and Chuang 2010].

2.1 Qubits and Gates

Whereas classical computers manipulate bits in order to solve a certain computational task, their quantum counterparts operate on so-called quantum bits, or *qubits*. A qubit is a two-level quantum system, i.e., a system which can be in two distinguishable states. An example would be the ground and first excited state of an ion, where we can denote the ground state by 0 and the first excited state as 1.

The principle of *quantum superposition* states that a single qubit can be in a complex superposition of its two levels. This means that there are two complex numbers associated with the quantum state of a qubit: the contribution from the 0-state and another one from the 1-state. Let us denote these two complex numbers by α_0 and α_1, respectively, where we require that $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Given a qubit in a quantum state which is described by these two values, the probability of observing the qubit in state 0 or 1 is $|\alpha_0|^2$ or $|\alpha_1|^2$, respectively. Note that the normalization condition above ensures that the two probabilities sum up to 1.

If we add a single qubit to a system of $n - 1$ qubits, the resulting system must be described in general using twice as many complex values due to *quantum entanglement.* Two subsystems being entangled means that one cannot write down the state of the entire system as a product state of the two subsystems. As a result, operations that act on one subsystem (such as measurement) may have nontrivial effects on the other. The state of an n-qubit quantum computer can be described using 2^n complex amplitudes that correspond to the contributions stemming from the all-zero state, the all-zero state but where the last bit is 1, up to the all-one state. We denote the corresponding amplitudes by $\alpha_{0\cdots00}, \alpha_{0\cdots01}, \ldots, \alpha_{1\cdots11}$ and, as a simplification, we interpret the indices as a number

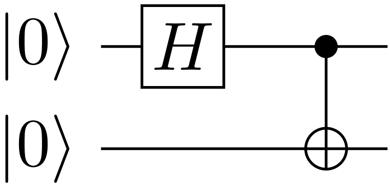


Fig. 2. Quantum circuit example: Each line represents a qubit and operations are drawn as boxes (e.g., the Hadamard operation H) or other symbols such as the controlled NOT or CNOT, which is depicted as $\oplus$ connected and attached to the filled circle on the control qubit. Time advances from left to right.

written in binary format, allowing to write $\alpha_0, \alpha_1, ..., \alpha_{2^n-1}$. When reading out, or *measuring*, all n qubits, the probability of observing the binary representation of i is given by $|\alpha_i|^2$. Once observed, the entire quantum system collapses onto the observed outcome, meaning that $\alpha_i = 1$ and $\alpha_j = 0$ for all $j \neq i$. In particular, repeated measurement results in the same answer.

When dealing with quantum systems, one typically employs the so-called *Dirac notation*, where state vectors correspond to so-called *kets* that are denoted by $|\cdot\rangle$, e.g., $|\psi\rangle$. Continuing the example from above, let $|\psi\rangle$ denote the quantum state of an n-qubit quantum computer. Using $|i\rangle$ with $i \in \{0, 1, ..., 2^n - 1\}$, we can write

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \ ,$$

with $\alpha_i \in \mathbb{C}$ the contribution from the $|i\rangle$-state and $\sum_i |\alpha_i|^2 = 1$. As above, the binary representation of i lets us determine the value of each of the n qubits in the i-th basis state $|i\rangle$. We note that the set $\{|i\rangle, i \in \{0, ..., 2^n - 1\}\}$ is called the *computational basis* in the quantum computing literature.

Due to the relationship between amplitudes and probabilities, all operations on qubits, so-called *quantum gates*, must preserve inner-products. As a result, quantum gates must be *unitary*, which means that for a quantum gate U,

$$U^\dagger U = UU^\dagger = \mathbb{1} \ ,$$

where $U^\dagger$ denotes the Hermitian adjoint of U. Note that this also implies that all quantum gates must be reversible and, therefore, that only reversible operations can be implemented using quantum gates. Such unitary operations may also be applied *controlled* on another qubit, meaning that they are applied if the control qubit is 1. Formally, the controlled version of U is

$$U^c := |0\rangle\langle 0| \otimes \mathbb{1} + |1\rangle\langle 1| \otimes U \ ,$$

where $|c\rangle\langle c|$ is the projector onto the subspace in which the control qubit has the value $c \in \{0, 1\}$ and $\otimes$ denotes the tensor product. Since the control qubit may be in a superposition, the state after applying U^c is in a superposition of having and not having applied U.

2.2 Quantum Programs

A quantum program is a classical program that, in addition to classical instructions, executes so-called *quantum circuits* on a quantum co-processor. In a quantum circuit, each qubit is represented as a horizontal line and quantum gates are denoted by boxes or other symbols on these lines, with time moving from left to right. See Fig. 2 for an example. It consists of a Hadamard gate H and a controlled NOT or CNOT gate. The CNOT is drawn as a NOT gate (denoted by $\oplus$) that is connected to a filled circle on the control qubit.

Definition 2.1. Quantum instruction. Let $O\,|q_1, ..., q_k\rangle$ denote a *quantum instruction.* It consists of an operation O and a k-tuple of qubits $(q_1, ..., q_k)$, where the operation may be a quantum gate or a classical instruction (allocation, deallocation, measurement).

Every circuit consists of the following 4 steps:

(1) Allocate n qubits in state $|0\rangle^{\otimes n} := |0 \cdots 0\rangle$ (n zeros)
(2) Apply quantum gates to these qubits
(3) Measure some or all of the qubits
(4) Deallocate measured qubits

Upon completion, the quantum co-processor returns a set of classical bits, the so-called *measurement results.* Depending on these results, the classical processor may then provide further quantum circuits to evaluate in order to solve the computational problem at hand. At the end of the entire quantum program, all qubits are deallocated again.

Since qubits are an extremely scarce resource, it is crucial to keep the number of allocated qubit minimal at any given point throughout the circuit and to deallocate all qubits which are no longer in use. Using the principle of deferred measurement [Nielsen and Chuang 2010], this means that as soon as the last operation on a given qubit has finished, the qubit can be measured and then freed for further use in the ongoing computation.

3 QUANTUM PROGRAM OPTIMIZATION USING ASSERTIONS

In this section, we introduce the basic idea of our assertion-based optimization methodology, followed by a proof of its correctness.

3.1 Entanglement Description Assertions for Quantum Program Optimization

Our goal is to use the knowledge of how subsystems are entangled for the purpose of quantum program optimization. To this end, we introduce the concept of *entanglement description assertions* that capture this information.

In particular, we introduce a formalism to describe the entanglement between qubits of the quantum computer throughout the execution of the quantum circuit. This entails statements that assert *entanglement descriptions* (to be defined next), that is, statements of the form

$$\text{“}\mathtt{q} == f(\mathtt{q},\ \mathtt{r})\text{”},\ \text{“}\mathtt{q} \geq f(\mathtt{q},\ \mathtt{r})\text{”},\ \text{etc.,}$$

where q and r refer to quantum registers and f is a function of two registers returning one register of bits. Since q and r refer to quantum registers, they may be in superposition and entangled with other qubits in the system. The following definition assigns a precise meaning to these *entanglement description assertions* with respect to the state vector of the entire n-qubit quantum computer,

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i\,|i\rangle\ .$$

Definition 3.1. Entanglement description assertion. Let $\square_{\text{cmp}}$ denote a comparison operator, $f : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^k$ a function on $k + m$ bits returning k bits, and let q, r be quantum registers consisting of k and m qubits, respectively. The *entanglement description assertion* $A(q,r) =$ q $\square_{\text{cmp}}\, f(\mathtt{q},\ \mathtt{r})$ on the n-qubit quantum state $|\psi\rangle$ asserts that

$$\forall i \in \{0, ..., 2^n - 1\} : (\,|\alpha_i| > 0 \implies A(\mathscr{Q}(i), \mathscr{R}(i))\,)\ ,$$

where the functions $\mathscr{Q} : \{0,1\}^n \to \{0,1\}^k$ and $\mathscr{R} : \{0,1\}^n \to \{0,1\}^m$ extract the bits corresponding to the quantum registers q and r, respectively, from the n-bit index i of the computational basis state $|i\rangle = |i_{n-1}, ..., i_0\rangle$.

With this definition in place, let us revisit the $|0\rangle$-control qubit example from the previous section and cast it as an *entanglement description assertion.*

Example 3.2. To express that a control qubit (denoted by $|c\rangle$) is in a definite state $|0\rangle$, let $f(\cdot,\cdot)=0$ and $\square_{\text{cmp}}$ be the equals comparison operator in the above definition. Then $A(c,\cdot) = (\texttt{c == 0})$. For the corresponding state $|\psi\rangle$, this means that $\alpha_i = 0$ whenever i corresponds to a state where the control qubit is 1. As a result, the action of a controlled gate U^c on $|\psi\rangle$ is always trivial.

This shows that such assertions can be used to express knowledge about qubits that are in a definite state. This piece of information can, when combined with classical constant-folding, be used for optimization. However, in order to do so in a more general setting, the optimizer also needs information which specifies the conditions for an operation to be trivial. We call this information *triviality conditions.*

Definition 3.3. Triviality condition. Let $A(q,r)$ be an entanglement description assertion on the quantum state $|\psi\rangle$. $A(q,r)$ is a *triviality condition* of a quantum operation U if

$$A(q,r) \implies U|\psi\rangle = |\psi\rangle \ ,$$

meaning that U acts as the identity if $A(q,r)$ is satisfied by $|\psi\rangle$.

Example 3.4. Continuing the $|0\rangle$-control qubit example, the triviality condition of the controlled unitary U^c would read $\{c == 0\}$ and, if this is satisfied as in the previous example, U^c may be removed from the circuit.

Therefore, using these two definitions, we can describe and carry out classical constant-folding. In order to see that this approach is strictly more powerful than classical constant-folding, consider the following example.

Example 3.5. Let $|\psi\rangle$ denote the quantum state of a two-qubit quantum computer. Initially, $|\psi\rangle = |00\rangle$ and our quantum program consists of two operations: 1) Prepare a Bell-pair and 2) swap the two qubits by applying a Swap gate. The Bell-pair preparation routine has $\{q_0 == 0, q_1 == 0\}$ as preconditions and ensures that $\{q_0 == q_1\}$ as a postcondition. In particular, given that the preconditions are satisfied, the Bell-pair preparation circuit in Fig. 2 transforms the state $|00\rangle$ to

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \ ,$$

The amplitudes of this quantum state are $\alpha_{00} = 1/\sqrt{2}$, $\alpha_{01} = \alpha_{10} = 0$, and $\alpha_{11} = 1/\sqrt{2}$. It is easy to check that the postcondition holds, i.e., that

$$\forall i \in \{0,1,2,3\} : |\alpha_i| > 0 \implies \{i_0 == i_1\},$$

where i_0 and i_1 denote the 0th and 1st bit of i, respectively. Since swaps are trivial if $q_0 == q_1$, which is satisfied by the state above, the Swap gate can be removed from the circuit. We note that the same reasoning applies if the Hadamard gate in the Bell-pair preparation circuit is replaced by an arbitrary rotation gate. In this case, $\alpha_{01} = \alpha_{10} = 0$ still holds, and so the Swap gate may be removed.

Since the quantum state in the example above is in a superposition, it is clear that regular constant-folding cannot successfully perform this optimization. Using our entanglement description assertions, however, this becomes feasible. This shows that optimization using entanglement description assertions is strictly more powerful than classical constant-folding.

3.2 Correctness of Our Methodology

We now prove that optimizations based on entanglement description assertions and triviality conditions leave the action of the overall quantum program invariant.

Theorem 3.6 (Correctness). *Let C be a quantum circuit that gets sent to a perfect*[1] *quantum device during execution of the quantum program $\mathcal{P}$. Applying assertion-based optimization to C will not change the output of $\mathcal{P}$.*

Proof. Let $A(q, r)$ denote an entanglement description assertion on the state $|\psi\rangle$ of the quantum device at a given point during the execution of C and let U be the next subroutine to be executed. Moreover, let $A'(q, r)$ be a triviality condition of U such that

$$A(q,r) \implies A'(q,r). \tag{1}$$

Then, applying U is equivalent to

$$\begin{aligned} |\psi\rangle \mapsto U|\psi\rangle &= \sum_{i=0}^{2^n-1} \alpha_i U|i\rangle \\ &= \sum_{i:A(\mathcal{Q}(i),\mathcal{R}(i))} \alpha_i U|i\rangle + \sum_{i:\neg A(\mathcal{Q}(i),\mathcal{R}(i))} \alpha_i U|i\rangle \end{aligned}$$

We know that $\neg A(\mathcal{Q}(i), \mathcal{R}(i)) \implies \alpha_i = 0$, since $|\psi\rangle$ satisfies $A(q, r)$. Thus,

$$U|\psi\rangle = \sum_{i:A(\mathcal{Q}(i),\mathcal{R}(i))} \alpha_i U|i\rangle .$$

Now, for all i in this superposition, $A'\left(\mathcal{Q}(i), \mathcal{R}(i)\right)$ holds due to (1) and U acts as the identity, i.e.,

$$\forall i : A\left(\mathcal{Q}(i), \mathcal{R}(i)\right) \implies U|i\rangle = |i\rangle ,$$

which lets us conclude that

$$U|\psi\rangle = \sum_{i:A(\mathcal{Q}(i),\mathcal{R}(i))} \alpha_i |i\rangle = |\psi\rangle .$$

Removing U from C does thus not affect $|\psi\rangle$ and, in particular, the final outcome of running $\mathcal{P}$ will not change by performing this optimization. Furthermore, our methodology will not modify the circuit if (1) does not hold. □

4 FORMALIZATION AND GENERALIZATION

In this section, we formalize the pre- and postconditions that are necessary to handle all examples in this paper, including the practical examples that will be introduced in the next section. Furthermore, we introduce a generalization of our methodology that is strictly more powerful.

4.1 Formalization of Our Basic Methodology

In order to formalize the basics of our methodology, we first define the pre- and postconditions of the quantum subroutines that are required for our examples in Table 1, where $X(q)$ denotes application of a Pauli-X [Barenco et al. 1995] gate to qubit q.

From the pre- and postconditions of the Swap operation, it is also apparent that a Swap is trivial if $q_i == q_j$; a fact that we already used in Example 3.5.

[1] Since circuit optimization may improve, e.g., success probability for a real device, we assume a hypothetical, perfect device.

Table 1. Pre- and postconditions of the quantum subroutines that are required to optimize our examples.

Operation	Preconditions	Postconditions
q=Alloc(n)	$\{q = \emptyset; n \in \mathbb{N}\}$	$\{q = \|0\rangle^{\otimes n}\}$
Dealloc(q)	$\{q = \|0\rangle^{\otimes n}\}$	$\{q = \emptyset\}$
Swap(q_i,q_j)	$\{q_i = A, q_j = B\}$	$\{ q_i = B, q_j = A\}$
X(q)	$\{q = A, A \in \{0, 1\}\}$	$\{ q = A \oplus 1\}$

In addition to the pre- and postconditions above, we require a formal description of the control modifier, which turns a given quantum subroutine U into its controlled version U^c, where c refers to the control qubit. The postcondition corresponding to

$$control(U)(c, q)$$

is $\{q = U^c |\psi\rangle\}$, where U^c is U on the subspace where $c = |1\rangle$ and $U^c = \mathbb{1}$ on the subspace where $c = |0\rangle$, and $|\psi\rangle$ denotes the state of the q-register before applying $control(U)$.

The pre- and postconditions of higher-level subroutines may be defined in a similar fashion. However, when combined, the pre-/postconditions above are sufficient for our examples. E.g., combining the control modifier with the NOT or Pauli X gate allows us to optimize the Bell-pair example where, after an initial Hadamard gate H [Barenco et al. 1995] on $|00\rangle$, the controlled NOT gate was applied as follows

$$H_1 |0\rangle |0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) |0\rangle \overset{CNOT}{\mapsto} \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) .$$

Since the controlled NOT gate flips the qubit in $|0\rangle$ if the control qubit is one, we immediately get the postconditions for the two qubits q_0 and q_1

$$\{q_1 = X^{q_0} |0\rangle\} \implies \{q_1 == q_0\} ,$$

by combining the pre- and postconditions of the control modifier and the Pauli X gate. Together with the triviality condition of the Swap gate acting on two qubits q_i and q_j,

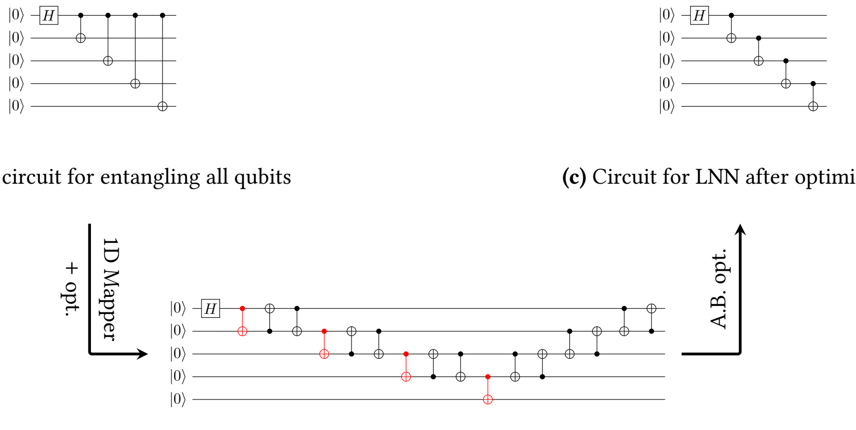
$$\{q_i == q_j\} ,$$

we can again remove the Swap gate from the circuit of the Bell-pair example. Similarly, the pre- and postconditions for CNOT can be used to identify the optimization opportunity in the following example.

Example 4.1. In addition to circuit optimizations at the logical level, entanglement description assertions and triviality conditions can be used to optimize the circuit for a specific target architecture. Consider the compilation steps outlined in Fig. 3. After mapping the circuit in **(a)** to a linear nearest-neighbor connectivity with additional optimizations to cancel intermediate partial Swap chains results in the circuit **(b)**. As before, we can employ our assertion-based optimizer to remove trivial CNOT gates using the fact that after each red CNOT gate acting on q_i and q_{i+1}, it holds that $q_i == q_{i+1}$. The optimized circuit is shown in Fig. 3**(c)**.

4.2 Generalized Optimization Methodology

By generalizing the basic methodology above, we can greatly increase its optimization capabilities. So far, our optimizer considers single gates at any given point together with all available postconditions of previously executed subroutines. For each such gate, it then determines whether it can be removed from the circuit without altering its output. The generalized strategy considers multiple gates and checks whether the supplied postconditions allow to deduce that the combined action of



(a) Original circuit for entangling all qubits

(c) Circuit for LNN after optimization.

(b) Circuit for LNN before A.B. opt.

Fig. 3. Optimizing a chain of CNOTs for a linear nearest-neighbor (LNN) architecture such as the 9-qubit chip by Google [Kelly et al. 2015] by employing the pre- and postconditions of CNOT gates. The benefit of our assertion-based optimization (A.B. opt.) can be seen clearly when comparing the circuits in **(b)** and **(c)**: No extra CNOTs due to Swaps [Kutin et al. 2007] are necessary in **(c)**, resulting in much lower gate count and circuit depth.

these gates is trivial, in which case all of these gates can be removed from the circuit. In order to properly introduce our generalized methodology, we first require a few definitions.

Definition 4.2. Set of control qubits. For an instruction $U\,|q_1, ..., q_k\rangle$ acting with a (unitary) gate U on k qubits, a set of qubits $\mathcal{S} \subset \{q_1, ..., q_k\}$ is called a *set of control qubits* if there exists a sequence of Swap gates $s_1, ..., s_t$ acting on pairs from $\{q_1, ..., q_k\}$ and a unitary U' such that with S denoting the unitary which performs $s_1, ..., s_t$, the following three statements hold.

(1) $SUS^\dagger = (\mathbb{1} - |1 \cdots 1\rangle \langle 1 \cdots 1|) \otimes \mathbb{1} + |1 \cdots 1\rangle \langle 1 \cdots 1| \otimes U'$
(2) the sequence of Swaps $(s_1, ..., s_t)$ permutes $(q_1, ..., q_k)$ such that the first $|\mathcal{S}|$ qubits of the resulting tuple are in $\mathcal{S}$
(3) $\mathcal{S}$ is the largest such set.

For instructions featuring a non-unitary operation (measurement, allocation, deallocation), the set of control qubits is empty.

We note that there may be multiple distinct sets of control qubits for a given instruction (as in the following example). For instructions where multiple choices exist, we choose a set of control qubits once and keep it invariant throughout the optimization process.

Example 4.3. As an example of an instruction where multiple choices exist for the set of control qubits, consider the Z^c operation applied to $|q_1 q_0\rangle$, where Z acts with a (-1)–phase on $|1\rangle$ and leaves $|0\rangle$ invariant. It is easy to check that

$$\begin{aligned} Z^c &= |0\rangle \langle 0| \otimes \mathbb{1} + |1\rangle \langle 1| \otimes Z \\ &= \mathbb{1} \otimes |0\rangle \langle 0| + Z \otimes |1\rangle \langle 1| \ , \end{aligned}$$

since for Z^c to be nontrivial, both qubits need to be in $|1\rangle$. Either qubit can thus be chosen to be the control qubit and, thus, the set of control qubits is not unique.

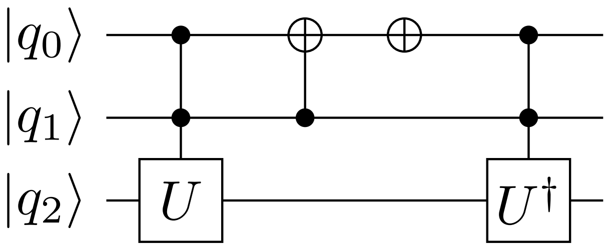


Fig. 4. Simple example of our multi-gate optimization methodology: The first gate is applied if and only if the last gate is applied (irrespective of the input state $|q_2, q_1, q_0\rangle$). Since the U gate is the inverse of $U^\dagger$, we can cancel the two doubly-controlled gates.

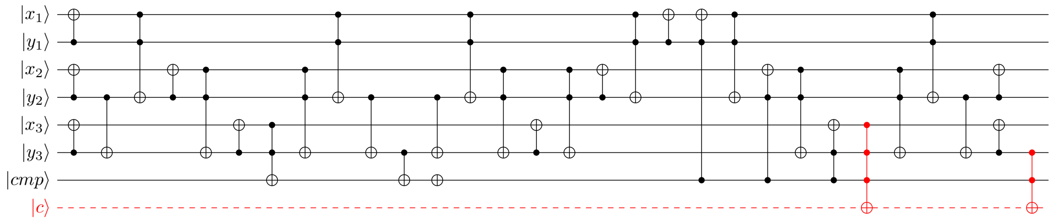


Fig. 5. Three-qubit example of a modular adder subroutine which performs the modular reduction. It consists of a comparison, the result of which is stored in the qubit $|cmp\rangle$, and a conditional subtraction. In this setting, our generalized methodology is able to deduce that the two red multi-controlled NOT gates can be canceled, allowing to completely remove the carry qubit $|c\rangle$. In a modular multiplier, n qubits would be saved since qubit reuse is not generally possible without uncomputation [Bennett 1973].

Definition 4.4. Target qubit. A qubit q in an instruction $U|..., q, ...\rangle$ is called a *target qubit* if it is not in the set of control qubits of the instruction $U|..., q, ...\rangle$.

Definition 4.5. Target-successive instructions. Two instructions I_1, I_2 with identical target qubits are called *target-successive* if no other instructions are scheduled to be executed between I_1 and I_2 that involve the target qubits in a way that does not commute with neither I_1 nor I_2.

Our generalized methodology considers $M \geq 1$ target-successive instructions at once, where all M instructions have the same t target qubits and arbitrary controls. Ignoring the control qubits, let $U_1, ..., U_M$ denote the t-qubit gate matrices of these instructions. An optimization can be performed, for example, if

$$U_M \cdots U_1 = \mathbb{1}_{2^t \times 2^t}$$

and the postconditions on the control qubits are such that either all or none of the gates get executed. A simple example with $M = 2$ and $t = 1$ is depicted in Fig. 4, where the two doubly-controlled gates can be canceled using this reasoning.

We now give a practical example where our multi-gate optimization strategy performs better than the single-gate methodology discussed thus far.

Example 4.6. Consider a circuit that performs addition modulo a quantum number N (stored in another quantum register), i.e.,

$$|a\rangle\,|b\rangle\,|N\rangle \mapsto |(a+b) \bmod N\rangle\,|b\rangle\,|N\rangle \ .$$

A possible implementation is to first perform the regular addition, followed by a modular reduction if the result is greater than $|N\rangle$. Since we only subtract N if $(a+b) \geq N$, the result will always be non-negative and, as a consequence, the final carry qubit will always be zero and it can thus be removed from the subtraction circuit. When using the addition circuit by Takahashi et al. [Takahashi et al. 2010], the optimizer needs to remove the two red multi-controlled NOT gates in Fig. 5 which act on the carry qubit in order to exploit this optimization opportunity. Neither of these gates is trivial on its own, but in this setting, either both or none of the two gates are triggered. As a result, this optimization can only be performed using our generalized approach. The achieved reduction in circuit width and depth can be found in Section 7, which discusses the results obtained using our implementation.

5 PRACTICAL EXAMPLES

Example 3.5 illustrates that optimization using entanglement description assertions is more powerful than classical constant-folding. Furthermore, Example 4.6 shows that our generalized methodology is strictly more powerful than regular assertion-based optimization. In this section, we discuss several practical examples that can be optimized using entanglement description assertions, but not using existing approaches for quantum circuit optimization. We mainly consider subroutines for quantum arithmetic, which is essential for most applications.

Perhaps surprisingly, most of the quantum gates required to run Shor's algorithm for factoring [Shor 1994] are due to the evaluation of modular exponentiation. In contrast to their classical counterparts, quantum computers must evaluate such classical functions on a superposition of inputs. Because the input is in a superposition, these functions cannot simply be evaluated on a classical computer. This would require reading out the state of the system, which would collapse the superposition and, thus, destroy any quantum speedup. Rather, these functions have to be implemented in terms of quantum gates in order to run them directly on the quantum computer. Further examples where the evaluation of such classical functions on a quantum computer is necessary are 1) the HHL algorithm for solving linear systems of equations, which requires computing the reciprocal [Harrow et al. 2009] and 2) certain algorithms for solving quantum chemistry problems: Babbush et al. [Babbush et al. 2016] have reduced the asymptotic runtime of a chemistry simulation algorithm by computing the entries of the Hamiltonian on-the-fly. This involves evaluating the Coulomb potential and various other mathematical functions which, e.g., describe the chosen orbitals.

In order to enable execution of such classical functions on a quantum computer, one may start by implementing subroutines for basic arithmetic such as addition and multiplication [Haener et al. 2018] in terms of quantum gates. These modules can then be combined to enable evaluating polynomials and further higher-level mathematical functions. We use the resulting subroutines as benchmarks and show how our methodology is able to reduce the quantum resource requirements.

5.1 Floating-Point Arithmetic Subroutine

As a first example in this section, we consider a subroutine that is omnipresent in floating-point arithmetic, namely that of *renormalization*. Renormalization is used during floating-point computations in order to bring intermediate results back into proper floating-point form. This can be achieved using two subroutines: The first subroutine determines the position p of the first nonzero bit of the mantissa. The second subroutine then shifts the mantissa to the left by the output of the first subroutine. A quantum circuit which determines the position of the first nonzero bit is shown in Fig. 6 and a circuit which shifts the mantissa $|x\rangle$ by $|p\rangle$ positions is depicted in Fig. 7. In order for the shift circuit to work properly for any input, it must allocate $2^{n_p}-1$ extra work qubits in order to catch the overflow from the shifted $|x\rangle$, where n_p is the number of qubits in the position register

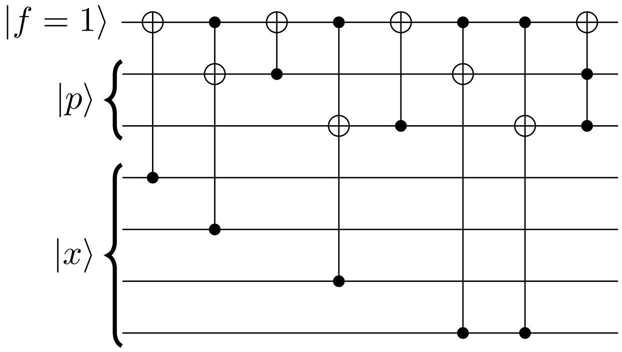


Fig. 6. Example of a circuit which finds the first nonzero bit of $|x\rangle$ and stores its position in $|p\rangle$ where $|x\rangle$ is a 4-qubit register and the position register $|p\rangle$ consists of two qubits [Haener et al. 2018]. The flag qubit $|f\rangle$ is one as long as the first one has not been found.

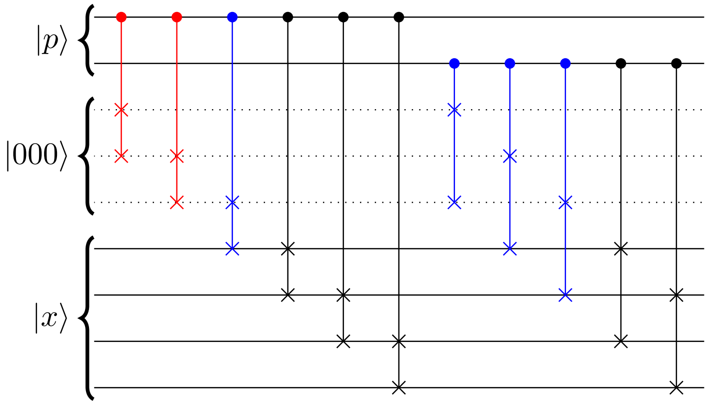


Fig. 7. Optimization of the shift circuit that can be performed if $|p\rangle$ contains the position of the first nonzero bit. Red Fredkin gates can be removed via regular constant-folding. Blue Fredkin gates can be removed using the postconditions of the subroutine depicted in Fig. 6 . As a result, all $2^{n_p} - 1$ work qubits can be eliminated (dotted lines).

$|p\rangle$. However, in the case where the input to the shift circuit gets initialized by the circuit which determines the position of the first one, such an overflow never occurs. As a result, the $2^{n_p} - 1$ work qubits can be eliminated from the combined circuit.

Identifying this optimization opportunity in the complete program is nontrivial and without some description of the action of gates or entire subroutines, such an optimization becomes completely infeasible for large circuits (as it would require simulation thereof for all inputs). We thus introduce a notion of how gates and subroutines interact by providing appropriate entanglement description assertions.

For this concrete example, consider the postcondition of the subroutine which determines the position pos of the first nonzero bit of $|x\rangle$. It asserts that the first pos qubits of x are zero, i.e.,

$$\forall i \in 0..\texttt{pos-1} : \texttt{x[i] == 0} \,,$$

where pos and x are entangled quantum variables. We can express this equivalently as an entanglement description assertion with

$$A_{FO}(x,p) = (x < 2^{n-p}) ,$$

where x is interpreted as an integer with x_0 as the most-significant bit (MSB) and p corresponds to the position register pos from above with p_0 being the least-significant bit (LSB). Using this postcondition, we now optimize the circuit in Fig. 7, which achieves the desired shift. Clearly, the red controlled Swap gates – also known as Fredkin gates – can be removed since they act on newly allocated qubits which are zero (the postcondition of qubit-allocation says that $q = |0\rangle^{\otimes n}$). The left-most blue Fredkin gate is a Swap gate controlled on the 0-th bit of $|p\rangle$ and thus acts trivially if $p_0 = 0$. Furthermore, the Swap itself is trivial if $x_0 = 0$ because all ancilla qubits are still in $|0\rangle$. Combining these two triviality conditions of the controlled Swap gate with the postcondition above yields that the blue Fredkin gate may act nontrivially only if

$$(p > 0) \wedge (x < 2^{n-p}) \wedge (x_0 \neq 0) ,$$

where x_0 denotes the MSB of the n-qubit register x. Clearly, these conditions cannot hold simultaneously and, as a result, the first blue Fredkin gate in Fig. 7 can be removed. Combining the postconditions of the Fredkin gates with $A_{FO}(x,p)$ yields a new assertion with

$$\begin{aligned} A_{new}(x,p) &= (2^{-p_0}x < 2^{n-p}) \\ &= (x < 2^{n-p+p_0}) , \end{aligned}$$

because if the first bit of the position register p_0 is one, we have just shifted all of x by one position. Since we successfully removed the first blue Fredkin gate, we can employ regular constant-folding to cancel the second blue Fredkin gate as well (all ancilla qubits are still in $|0\rangle$). For the final two blue Fredkin gates, note that they act nontrivially only if

$$(p_1 \neq 0) \wedge ((x_0 \neq 0) \vee (x_1 \neq 0)) .$$

From which we can use $p_1 \neq 0$ and combine it with the updated postcondition with a case-distinction on p_0: If p_0 is zero, then $p \geq 2$ and if p_0 is one, we have that $p \geq 3$ and that there is a shift of $+1$ in the exponent of the updated postcondition. Thus, in both cases,

$$x < 2^{n-2} ,$$

and hence, the two most-significant bits x_0, x_1 of x must be zero. The action of the remaining two blue Fredkin gates is therefore always trivial and they can also be removed from the circuit. Finally, since none of the allocated overflow qubits will be used anymore throughout the computation (as their content is always trivial in this application), they will eventually get deallocated without any operations having acted on them. It is then a simple local optimization to cancel allocations with subsequent deallocations, allowing to reduce the width of the resulting circuit by $2^{n_p} - 1$ qubits, as desired.

5.2 Fixed-Point Arithmetic Subroutine

Similar optimization opportunities arise when using a fixed-point representation. As an example, consider the evaluation of a function using a range reduction, e.g., evaluating the function $f(x) = \sqrt{x}$ for $x \in [0, 2)$.

One approach to evaluate $f(x)$ is to approximate the function on the interval $[1, 2)$ by a polynomial. We can then perform range reduction for every input x to $y := x \cdot 2^k \in [1, 2)$, for $k \in \mathbb{N}$, evaluate the polynomial for y and then use that

$$f(x) = \sqrt{x \cdot 2^k} 2^{-k/2} ,$$

where both multiplications by powers of two can be implemented using shifts and an additional multiplication by $\sqrt{2}$ for the $k/2$ exponent with an odd k. The function $f(x)$ can thus be evaluated on a quantum computer as follows:

1. Determine the position of the first non-zero qubit of x (starting from the most-significant bit)
2. Shift all bits of the fixed-point number such that the position is aligned with the binary point (and the number is now in the interval $[1,2)$)
3. Evaluate the polynomial approximating $f(x)$ on $[1,2)$ and store the result in a new quantum register
4. On the result register, undo half of the shift and multiply by $\sqrt{2}$ for odd shifts

We know that x was shifted by k positions in step 2. Therefore, the last k qubits of x are zero (where k is a quantum integer). These qubits can be used as work qubits when undoing half of the shift on the result register and our assertion-based optimizer can thus save an entire quantum fixed-point register.

More specifically, after having shifted the qubits of x toward the MSB, we have the same entanglement description assertion as in the previous example, $A_{FO}(x,p)$, between the x register and the position register. Now, the library implementation of the shift circuit should be able to use these "free" qubits of x as scratch space when shifting the output of the polynomial evaluation subroutine. This can be achieved through weakening of the preconditions on the work qubits of the shift circuit that catch potential overflow: Instead of requiring $2^{n_p}-1$ qubits in $|0\rangle$, it is sufficient that the first k qubits be zero, where k is a quantum integer denoting the distance of the shift. While this precondition can always be satisfied by allocating $2^{n_p}-1$ work qubits in $|0\rangle$, stating the weakened version allows the compiler to perform this optimization.

5.3 Integer Addition and Dirty Qubits

Recently, it was shown that the cost of an integer addition subroutine can be halved using the fact that the target qubit after an uncompute Toffoli gate is back in $|0\rangle$ [Gidney 2018]. Using the pre- and postconditions of allocation and deallocation, our optimization methodology can identify and exploit this optimization opportunity.

A similar optimization can be applied for subroutines that employ so-called *dirty qubits* [Barenco et al. 1995; Häner et al. 2017]. While such implementations use fewer (clean) work qubits, they typically cause an increase in circuit depth. Thus, when compiling the program for a specific architecture, the compiler should use as many clean qubits as possible in order to reduce this negative effect on the runtime. Because our assertion-based optimizer is able to identify when a dirty qubit gets mapped to a qubit that is actually clean, it can then optimize the resulting circuit.

For n-ary controlled NOT operations [Barenco et al. 1995], this translates to canceling Toffolis that are guaranteed not to be applied because one of the control qubits is in $|0\rangle$ (since it is a clean qubit). For an addition-by-constant circuit that uses dirty qubits, this translates to removing both the controlled inversions and a controlled incrementer [Häner et al. 2017, Fig. 5]. These automatic conversions from dirty to clean qubits allow savings of approximately 2× in the number of gates and, crucially, allow for more modularity when implementing libraries for quantum computing.

As a technical detail, note that gates can be removed both after allocation and before deallocation: By definition, a dirty qubit must be returned to its original state before deallocation and so the clean qubit will have been brought back to $|0\rangle$.

6 IMPLEMENTATION USING PROJECTQ AND Z3

In this section, we discuss our implementation of our optimization methodology. We implement our methodology using the ProjectQ software framework for quantum computing [Steiger et al.

2018]. ProjectQ features an extensible compiler framework, allowing to easily integrate custom compiler passes such as our optimization methodology.

For each quantum operation for which we would like to add nontrivial optimization capabilities using our approach, we add the corresponding post- and triviality conditions. Additionally, preconditions may be supplied which would allow to test the program for correctness. To add support for our generalized methodology, we only require the triviality condition of the control modifier, in addition to information which lets us determine whether a sequence of operations $U_1, ..., U_M$ acts as the identity. The latter is already available in ProjectQ.

We extend the definitions of several quantum gates in ProjectQ with the corresponding entanglement descriptions (both post- and triviality conditions). Specifically, we add member functions that use functionality from the Z3 Theorem Prover package [de Moura and Bjørner 2008] to express these conditions. Our custom compiler pass uses these member functions in combination with the Z3 solver in order to check whether certain operations are guaranteed to be trivial, in which case they can be removed.

While we do not elaborate on the details of the ProjectQ compilation framework, we point out that optimization and compilation is carried out during circuit generation time. As a result, all parameters of the circuit are already known. In particular, the lengths of all quantum registers are known since all classical inputs to the quantum program have been supplied. The circuit can thus be optimized specifically to the problem instance in question. Furthermore, this enables more powerful optimizations when employing our methodology because we do not require parametric proofs. It is of course theoretically possible to prove such statements by induction, however, there is only limited support in automatic theorem provers such as Z3 [de Moura and Bjørner 2008] due to the difficulty of, e.g., constructing appropriate induction rules [Bundy 1999]. Since all classical parameters have a definite value upon circuit generation, we can unroll quantified statements and thereby generate claims that are easier to prove.

As an example, we show how the definition of the ProjectQ Swap operation was altered in order to enable our optimization engine to carry out the optimizations discussed so far. The definition of `SwapGate` was extended by merely the following two member functions:

```
class SwapGate(SelfInverseGate):
    [...]
    def trivial_if(self, x1, x2):
        return (x1 == x2)

    def postconditions(self, x1, x2, y1, y2):
        return And(x1 == y2, x2 == y1)
```

Clearly, these are very minor modifications that provide exactly the information required: Postconditions and triviality conditions of the Swap gate. The `trivial_if` member function of every gate is invoked by the optimizer with one symbolic boolean variable for each target qubit of the gate (two in this case). The returned expression is negated and then added to the solver together with the expression ctrls_one = And(v[cqb$_1$], v[cqb$_2$], ...), which is true if and only if all variables v[cqb$_i$] corresponding to control qubits cqb$_i$ that are true / equal to one:

```
solver.push()
solver.add(And(ctrls_one, Not(cmd.gate.trivial_if(*target_vars))))
if solver.check() == unsat:
  ... # skip current operation
solver.pop()
```

where target_vars are the Z3 variables corresponding to the target qubits of the current gate before it is executed. If the solver finds a solution that satisfies all previous conditions and the negated conditions of trivial_if, the gate cannot be removed since it may have a nontrivial effect on the state of the quantum computer $|\psi\rangle$ at that point. If there is no such solution, on the other hand, this means that the gate is trivial and it can thus be removed from the circuit. After this triviality check, the conditions of the Z3 solver are updated according to the postconditions of the operation which hold irrespective of whether the gate was removed: For each target qubit, a new boolean Z3 variable is created and the postconditions member function of the gate relates the old variables (before applying the gate) to the new ones. In particular, operations are handled by adding two Z3 Implies(...) statements:

(1) The control qubit(s) being all ones implies that the new target variables are now related to the old ones via the postconditions function, i.e.,

```
Implies(ctrls_one, cmd.gate.postconditions(*(target_vars+
    new_target_vars)))
```

is added to the solver, where new_target_vars are the Z3 variables that correspond to the target qubits after applying the gate.

(2) The control qubit(s) not being all ones implies that the new target variables are equal to the old ones, i.e., for all i we add the expression

```
Implies(Not(ctrls_one), new_target_vars[i] == target_vars[i]))
```

to the solver.

If there are no control qubits, (1) and (2) are of the form

$$\{\texttt{true} \implies y = f(x)\} \text{ and } \{\texttt{false} \implies y = x\} ,$$

respectively and, therefore, are equivalent to stating that $y = f(x)$ holds after the gate has been applied, where f is given by the postconditions member function. As a technical detail, note that the ProjectQ Swap gate derives from SelfInverseGate, stating that the Swap operation is its own inverse. This information is useful for our generalized optimization approach, which is employed whenever the circuit buffer size of the optimizer exceeds a user-defined threshold. When this happens, the stored circuit is traversed in order to identify target-successive operations which may be removed from the circuit. For each such sequence of gates, the Z3 solver is used to determine whether there is an assignment to the control qubits that agrees with all previous postconditions and that causes $0 < m < M$ operations to be executed. If there is no such assignment, either all or none of these M operations are executed, meaning that they always act trivially. As a result, the entire sequence of gates can be removed from the circuit.

7 RESULTS

In this section, we report the results that were obtained using our prototype implementation of the proposed optimization methodology. We demonstrate the performance of our optimizer using three different quantum subroutines. The first subroutine performs floating-point mantissa renormalization, see Figs. 6 and 7, the second entangles a linear chain of qubits, see Fig. 3, and the third performs modular reduction, see Fig. 5, which is a subroutine that is used in constructing a modular adder.

For all subroutines, we compare two ProjectQ compiler setups—one which features a local optimizer capable of merging/canceling subsequent operations that act on the same qubits, and a second configuration which additionally contains our assertion-based optimizer. We choose {CNOT, X, H, S, T, $T^\dagger$} as the target gate set for both configurations. In order to compare these

Table 2. Optimizer comparison for the floating-point renormalization circuit for 8-, 16-, 32-, and 64-qubit floating-point, where $n_p = 3, 5, 8, 11$, respectively, and the mantissa contains $n - n_p - 1$ qubits. Our entanglement description based optimizer achieves a reduction in circuit area (width×depth) of up to $410\times$.

n	width	depth	optimized width	optimized depth	area reduction
8	15	235	7	107	$4.7\times$
16	47	922	15	388	$7.4\times$
32	287	5538	31	1080	$47.5\times$
64	2111	38903	62	3229	$410.2\times$

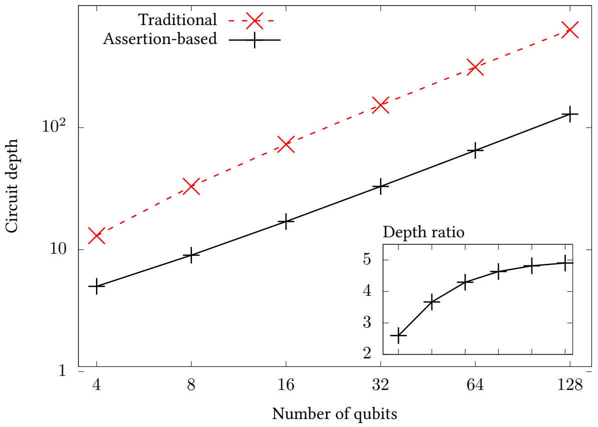


Fig. 8. Optimizer comparison for the entangling circuit on n qubits depicted in Fig. 3. The assertion-based optimizer achieves a $5\times$ improvement in circuit depth for large n.

different configurations, we use circuit width and depth as benchmark numbers. The circuit width corresponds to the maximal number of alive qubits at any point throughout the execution of the circuit. The *circuit depth* is equal to the delay of the circuit assuming that all quantum gates in the target gate set take unit time.

The comparisons can be found in Table 2 and Fig. 8 for the floating-point renormalization circuit and the entangling circuit, respectively. Both cases clearly demonstrate the benefits of our assertion-based optimizer, which is able to reduce the circuit area (width × depth) by a factor of up to $410\times$ and $5\times$ for the first and second circuit, respectively.

For the floating-point renormalization circuit, our optimizer is able to eliminate $2^{n_p} - 1$ ancilla qubits in addition to several Fredkin gates. Due to the elimination of Fredkin gates, the depth is also significantly reduced. We note that these savings are obtained without any prior manual optimizations such as limiting the maximal shift to the number of bits in the mantissa, as this describes a realistic scenario for software reuse.

For the entangling circuit, all CNOT gates resulting from swap operations can be removed when using the assertion-based optimization strategy (see Example 3.5 for more details). Therefore, the circuit depth would grow by $4(n-2)$ gates for $n \geq 2$ when turning off assertion-based optimization (see Fig. 3). The ratio between the resulting circuit depths for $n \geq 2$ is thus

$$\frac{4(n-2)+n}{n} = \frac{5n-8}{n} \overset{n\to\infty}{\longrightarrow} 5\,,$$

which agrees with the experimental results in Fig. 8 and constitutes an up to 5× improvement over state-of-the-art optimizers.

The modular reduction circuit, which is a subroutine for modular addition, is optimized by identifying a pattern similar (but more complex) to the one shown in Fig. 4. In this case, the target qubit is the carry qubit of the controlled subtraction and upon removing the two multi-controlled NOT operations, no operations on the carry qubit remain. As a result, our methodology removes this qubit from the circuit. Furthermore, our methodology achieves a slight depth reduction due to the removal of the two generalized Toffoli gates. In Shor's algorithm for factoring an n-bit number, n calls to such a modular reduction are required. The total savings would thus be equal to n qubits, where $n \sim 2000$ for practical applications.

8 SUMMARY AND FUTURE WORK

We have presented an optimization methodology that extends the scope of automatic circuit optimizations. In particular, our methodology carries out high-level optimizations across subroutine boundaries that are typically performed by humans, enabling more efficient code reuse. This is achieved by taking into account pre-, post-, and triviality conditions of all subroutines that get invoked by the quantum program that is being optimized.

Our generalized methodology currently performs optimizations if the overall action of a sequence of gates is trivial. Future work could address more general cases where, e.g., control qubits are in a state that only triggers subsets of these gates that, when combined, correspond to trivial operations. Additionally, symbolic computation on entanglement description assertions may be incorporated. This would allow to optimize iterative procedures such as the Newton-Raphson method which can be used to evaluate high-level arithmetic functions on a quantum computer [Cao et al. 2013]: For many such functions, the initial guesses can be chosen to be very simple (e.g., integer powers of two). The first iteration of a Newton-Raphson method may then be applied symbolically to the output of the initial guess routine. Such optimizations have been shown to yield significant resource savings when performed manually [Häner et al. 2018a]. Automating such procedures would thus result in the same benefits without the need for labor-intensive manual code optimization. Moreover, the focus of the present work lies on optimizing permutation-type subroutines. Future work could extend this to target phase-oracles by supporting pre- and postconditions in multiple bases.

ACKNOWLEDGMENTS

We thank the anonymous referees for their valuable comments and suggestions. This work was supported by Microsoft and by the Swiss National Science Foundation through the National Competence Center for Research NCCR QSIT.

REFERENCES

M. Amy, D. Maslov, and M. Mosca. 2014. Polynomial-Time T-Depth Optimization of Clifford+T Circuits Via Matroid Partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33, 10 (Oct 2014), 1476–1489. https://doi.org/10.1109/TCAD.2014.2341953

Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. 2013. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 6 (2013), 818–830. https://doi.org/10.1109/TCAD.2013.2244643

Ryan Babbush, Dominic W Berry, Ian D Kivlichan, Annie Y Wei, Peter J Love, and Alán Aspuru-Guzik. 2016. Exponentially more precise quantum simulation of fermions in second quantization. *New Journal of Physics* 18, 3 (2016), 033032. https://doi.org/10.1088/1367-2630/18/3/033032

Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. 1995. Elementary gates for quantum computation. 52 (03 1995).

CH Bennett. 1973. Logical reversibility of computation. *Maxwell's Demon. Entropy, Information, Computing* (1973), 197–204.

Alan Bundy. 1999. *The automation of proof by mathematical induction.* Technical Report.

Yudong Cao, Anargyros Papageorgiou, Iasonas Petras, Joseph Traub, and Sabre Kais. 2013. Quantum algorithm and circuit design solving the Poisson equation. *New Journal of Physics* 15, 1 (2013), 013021. http://stacks.iop.org/1367-2630/15/i=1/a=013021

Frederic T Chong, Diana Franklin, and Margaret Martonosi. 2017. Programming languages and compiler design for realistic quantum hardware. *Nature* 549, 7671 (2017), 180. https://doi.org/10.1038/nature23459

Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 337–340.

Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. 2012. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A* 86 (Sep 2012), 032324. Issue 3. https://doi.org/10.1103/PhysRevA.86.032324

Craig Gidney. 2018. Halving the cost of quantum addition. *Quantum* 2 (June 2018), 74. https://doi.org/10.22331/q-2018-06-18-74

Craig Gidney and Martin Ekerå. 2019. How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits. *arXiv preprint arXiv:1905.09749* (2019).

Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. 2013. Quipper: a scalable quantum programming language. In *ACM SIGPLAN Notices*, Vol. 48. ACM, 333–342. https://doi.org/10.1145/2499370.2462177

Daniel Große, Xiaobo Chen, Gerhard W Dueck, and Rolf Drechsler. 2007. Exact SAT-based Toffoli network synthesis. In *Proceedings of the 17th ACM Great Lakes symposium on VLSI.* ACM, 96–101.

Daniel Große, Robert Wille, Gerhard W Dueck, and Rolf Drechsler. 2009. Exact multiple-control toffoli network synthesis with SAT techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28, 5 (2009), 703–715.

Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing* (Philadelphia, Pennsylvania, USA) *(STOC '96).* ACM, New York, NY, USA, 212–219. https://doi.org/10.1145/237814.237866

Thomas Haener, Mathias Soeken, Martin Roetteler, and Krysta M. Svore. 2018. Quantum Circuits for Floating-Point Arithmetic. In *Reversible Computation*, Jarkko Kari and Irek Ulidowski (Eds.). Springer International Publishing, Cham, 162–174. https://doi.org/10.1007/978-3-319-99498-7_11

Thomas Häner, Martin Roetteler, and Krysta M. Svore. 2017. Factoring Using 2N + 2 Qubits with Toffoli Based Modular Multiplication. *Quantum Info. Comput.* 17, 7-8 (June 2017), 673–684. http://dl.acm.org/citation.cfm?id=3179553.3179560

Thomas Häner, Martin Roetteler, and Krysta M Svore. 2018a. Optimizing Quantum Circuits for Arithmetic. *arXiv preprint arXiv:1805.12445* (2018).

Thomas Häner, Damian S. Steiger, Krysta Svore, and Matthias Troyer. 2018b. A Software Methodology for Compiling Quantum Programs. *Quantum Science and Technology* 3, 2 (2018), 020501. https://doi.org/10.1088/2058-9565/aaa5cc

Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. 2009. Quantum Algorithm for Linear Systems of Equations. *Phys. Rev. Lett.* 103 (Oct 2009), 150502. Issue 15. https://doi.org/10.1103/PhysRevLett.103.150502

Matthew B. Hastings, Dave Wecker, Bela Bauer, and Matthias Troyer. 2015. Improving Quantum Algorithms for Quantum Chemistry. *Quantum Info. Comput.* 15, 1-2 (Jan. 2015), 1–21. http://dl.acm.org/citation.cfm?id=2685188.2685189

IBM. 2018. *QISKit.* https://qiskit.org

Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T. Chong, and Margaret Martonosi. 2014. ScaffCC: a framework for compilation and analysis of quantum computing programs. In *Proceedings of the 11th ACM Conference on Computing Frontiers.* ACM, 1. https://doi.org/10.1145/2597917.2597939

N Cody Jones, James D Whitfield, Peter L McMahon, Man-Hong Yung, Rodney Van Meter, AlÃąn Aspuru-Guzik, and Yoshihisa Yamamoto. 2012. Faster quantum chemistry simulation on fault-tolerant quantum computers. *New Journal of Physics* 14, 11 (2012), 115023. https://doi.org/10.1088/1367-2630/14/11/115023

Julian Kelly, R Barends, AG Fowler, A Megrant, E Jeffrey, TC White, D Sank, JY Mutus, B Campbell, Yu Chen, et al. 2015. State preservation by repetitive error detection in a superconducting quantum circuit. *Nature* 519, 7541 (2015), 66.

Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. 2013. Fast and efficient exact synthesis of single qubit unitaries generated by Clifford and T gates. *Quantum Information & Computation* 13, 7-8 (June 2013), 0607–0630. arXiv:1206.5236

Samuel A Kutin. 2006. Shor's algorithm on a nearest-neighbor machine. *arXiv preprint quant-ph/0609001* (2006).

Samuel A Kutin, David Petrie Moulton, and Lawren M Smithline. 2007. Computation at a distance. *arXiv preprint quant-ph/0701194* (2007).

Giulia Meuli, Mathias Soeken, and Giovanni De Micheli. 2018. SAT-based {CNOT, T} Quantum Circuit Synthesis. In *International Conference on Reversible Computation.* Springer, 175–188.

Yunseong Nam, Neil J Ross, Yuan Su, Andrew M Childs, and Dmitri Maslov. 2018. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information* 4, 1 (2018), 23. https://doi.org/10.1038/s41534-018-0072-4

Michael A Nielsen and Isaac Chuang. 2010. Quantum computation and quantum information.

Jennifer Paykin, Robert Rand, and Steve Zdancewic. 2017. QWIRE: a core language for quantum circuits. *ACM SIGPLAN Notices* 52, 1 (2017), 846–858. https://doi.org/10.1145/3093333.3009894

Markus Reiher, Nathan Wiebe, Krysta M. Svore, Dave Wecker, and Matthias Troyer. 2017. Elucidating reaction mechanisms on quantum computers. *Proceedings of the National Academy of Sciences* 114, 29 (2017), 7555–7560. https://doi.org/10.1073/pnas.1619152114 arXiv:https://www.pnas.org/content/114/29/7555.full.pdf

Peter W Shor. 1994. Algorithms for quantum computation: Discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*. IEEE, 124–134. https://doi.org/10.1109/SFCS.1994.365700

Robert S. Smith, Michael J. Curtis, and William J. Zeng. 2016. A Practical Quantum Instruction Set Architecture. *arXiv preprint arXiv:1608.03355* (2016).

Damian S. Steiger, Thomas Häner, and Matthias Troyer. 2018. ProjectQ: an open source software framework for quantum computing. *Quantum* 2 (2018), 49. https://doi.org/10.22331/q-2018-01-31-49

Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. 2018. Q#: Enabling Scalable Quantum Computing and Development with a High-level DSL. In *Proceedings of the Real World Domain Specific Languages Workshop 2018* (Vienna, Austria) *(RWDSL2018)*. ACM, New York, NY, USA, Article 7, 10 pages. https://doi.org/10.1145/3183895.3183901

Yasuhiro Takahashi, Seiichiro Tani, and Noboru Kunihiro. 2010. Quantum Addition Circuits and Unbounded Fan-out. *Quantum Info. Comput.* 10, 9 (Sept. 2010), 872–890. http://dl.acm.org/citation.cfm?id=2011464.2011476

Mingsheng Ying. 2012. Floyd–hoare Logic for Quantum Programs. *ACM Trans. Program. Lang. Syst.* 33, 6, Article 19 (Jan. 2012), 49 pages. https://doi.org/10.1145/2049706.2049708