Using Hoare logic for quantum circuit optimization

Thomas Häner
Institute for Theoretical Physics
ETH Zurich
Zurich, Switzerland
haenert@phys.ethz.ch

Torsten Hoefler
Computer Science
ETH Zurich
Zurich, Switzerland
htor@inf.ethz.ch

Matthias Troyer
Institute for Theoretical Physics
ETH Zurich
Zurich, Switzerland
troyer@phys.ethz.ch

Abstract

By employing quantum mechanical phenomena such as superposition, entanglement, and interference, quantum computers promise to perform certain computations exponentially faster than any classical device. Precise control over these physical systems 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 successfully 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 that arise due to entanglement. To address this insufficiency, we introduce an optimization methodology which employs Hoare triples in order to identify and exploit these optimization opportunities. We implement the optimizer using the Z3 Theorem Prover and the ProjectQ software framework for quantum computing and show that it is able to reduce the circuit area of our benchmarks by up to 5×.

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 [31]. For quantum chemistry applications, such optimizations have enabled a reduction in time complexity from $O(N^{11})$, where N is the number of spin orbitals, to $O(N^5)$ [17, 26, 30, 38]. Shor's algorithm for factoring integers [32], which offers a superpolynomial speedup over classical algorithms, has been optimized manually in a similar fashion, resulting in significant resource savings [4, 13, 24, 36].

Such improvements are crucial to the viability of using quantum computing for these tasks, especially considering the overhead due to quantum error correction [8] and the difficulty of engineering large-scale quantum computers.

In light of this, a host of software packages, programming languages, and compilers for quantum computing have been developed [9, 19, 20, 33–35]. In addition to providing the necessary layers of abstraction to facilitate software development, these packages include optimizing compilers. Inspired by previous work in the classical domain, these programs allow merging of quantum operations at various layers of abstraction [15, 34], e.g., merging of rotations that are applied successively to the same quantum bit (qubit), and using code annotations to identify patterns that are common in quantum computing, akin to pragma statements in classical computing [15, 34]. Further optimization opportunities can be created by employing a set of commutation relations [28] to reorder operations. In general, however, this approach incurs a cost that is exponential in the number of qubits that the reordered operations act upon. Furthermore, several methods have been developed for exact circuit synthesis with certain optimality guarantees [1, 10, 11, 23, 27]. However, these methods are not suitable for optimization of large quantum circuits.

Despite these efforts, most of the progress made in, e.g., the aforementioned quantum chemistry applications have been due to manual circuit optimization [17, 21] and the derivation and evaluation of superior error bounds [30]. This suggests that the capabilities of optimizing compilers may still be significantly improved.

To this end, one may first consider differences between manual and automatic optimization. For instance, in contrast to automatic methods, humans tend to harness 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 is often readily available in the form of Hoare triples [18], albeit for the purpose of verification.

In this paper, we thus consider the use of Hoare triples for optimization. Specifically, we use the postconditions of each subroutine in order to gather information about the state of the quantum computer after completion of the subroutine. The gathered information is then combined with conditions specifying the circumstances under which a given operation 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.

We develop a formalism which allows to express such conditions in code. We then demonstrate the benefits of our optimization methodology. More specifically, we compile several examples using our implementation which we construct by employing the Z3 Theorem Prover [7] and ProjectQ, which

is a software framework for quantum computing [15, 34]. When compared to the state of the art, our optimization methodology achieves reductions in circuit area of up to 5×. Specifically, it achieves a reduction of 2× for floating-point mantissa renormalization and 5× for mapping an entangling circuit to a 1D linear chain. In particular, the use of Hoare triples allows our methodology to perform certain optimizations that would typically be performed only by humans.

2 Qubits and gates

Before defining what quantum programs are, we provide some background on quantum computing. For a more in-depth treatment of the subject, we refer to the textbook by Nielsen and Chuang [29].

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 completely 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. An n-qubit quantum computer can be described using 2^n complex amplitudes which 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 written in binary format, allowing to write $\alpha_0, \alpha_1, \ldots, \alpha_{2^n-1}$. 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.

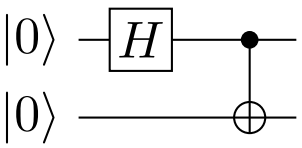


Figure 1. Quantum circuit example: Each line represents a qubit and operations are drawn as boxes (e.g., the Hadamard operation H) or other symbols such as the controlled NOT or CNOT, which is depicted as $\oplus$ connected and attached to the filled circle on the control qubit. Time advances from left to right.

When dealing with quantum systems, one typically employs the so-called *Dirac notation*, where state vectors correspond to so-called *kets* which 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 $i \in \{0, 1, \ldots, 2^n - 1\}$, 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, i.e., from the binary representation of i we can determine the value of each of the n qubits in the i-th basis state $|i\rangle$. We note that the set $\{|i\rangle, i \in \{0, \ldots, 2^n-1\}\}$ is called the *computational basis* in the quantum computing literature.

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

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

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.

3 Quantum programs

The machine model that we consider is a combination of a classical von Neumann architecture and so-called *quantum circuits* which get sent to the quantum co-processor for execution. 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. 1 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 3.1. 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 rare 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 [29], 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.

4 Hoare tiples and their use for optimization

Hoare logic [18] is a system which can be used to verify the correctness of a given program. To this end, every subroutine in the program is equipped with additional information, the so-called Hoare triple, consisting of preconditions, the function, statement or subroutine to execute, and the corresponding postconditions. This is written as

$$\{P\}F\{Q\}\ ,$$

where P denotes the preconditions, F the function to execute, and Q the postconditions.

From an abstract point of view, the given program can then be seen as a sequence of transition rules, where each such rule is given by a Hoare triple. If each such transition happens in a way that ensures the preconditions of the next transition, the Hoare triple of the entire program, taking the initial conditions of the first transition to the postconditions of the final Hoare triple is provably correct. Whether the specifications of pre- and postconditions actually agree with the implementation is a different issue that we shall not be concerned with in this work.

A first step toward optimization via Hoare triples is to provide multiple implementations that ensure identical postconditions. The most general implementation assumes minimal preconditions for the operation to be sensible. As a result, this implementation may perform worse than a less general implementation that is only valid if certain additional conditions are satisfied. These conditions may be added as preconditions to the less general implementation, allowing the compiler to chose between the two implementations automatically.

First, we provide two simple examples in the quantum domain for this type of optimization.

Example 4.1. A straightforward example is that controlled operations U^c can be removed if the control qubit $|c\rangle$ is known to be $|0\rangle$, or that the control can be removed and the operation U can be applied directly if the control qubit is in definite $|1\rangle$. Since implementing U^c is more expensive than implementing just U, either case yields an advantage. In terms of Hoare triples, this can be phrased as follows:

precondition: Instruction qubit(s) in $|c\rangle\,|\psi\rangle$
operation: Apply U^c
postcondition: Instruction qubit(s) in $|c\rangle\,U^c\,|\psi\rangle$

However, also the identity operation (which has zero implementation cost) ensures the same postcondition albeit with a more restrictive precondition:

precondition: Instruction qubit(s) in $|0\rangle\,|\psi\rangle$
operation: None
postcondition: Instruction qubit(s) in $|c\rangle\,U^c\,|\psi\rangle$

And similarly for the control qubit in definite $|1\rangle$.

Example 4.2. A more practical example is addition by a classical constant, i.e., the n-qubit mapping

$$|x\rangle \mapsto |x + c\rangle\ .$$

The compiler can either use the addition circuit by Häner et al. [13] which requires $O(n \log n)$ operations or, if extra n clean qubits in $|0\rangle$ are available, a full addition circuit such as the one by Takahashi et al. [37] which only consists of $O(n)$ operations. The latter amounts to performing

$$|x\rangle\,|0\rangle \mapsto |x\rangle\,|c\rangle \mapsto |x + c\rangle\,|c\rangle \mapsto |x + c\rangle\,|0\rangle\ .$$

The translation to Hoare triples is again straightforward: With the additional precondition that n qubits are available as work qubits, the $O(n)$ gate addition circuit ensures the output to be $|x + c\rangle$.

While optimizations mentioned thus far can be performed using Hoare triples, knowledge of the Hoare triple is not strictly necessary to do so. For example, simple constant-folding can be used to remove controlled gates, where the control qubit is in the computational basis state $|0\rangle$.

In what follows, we extend the optimization capabilities of compilers to handle cases that could not be optimized without the additional information provided by Hoare triples.

5 Compiler optimization via Hoare logic

In this section, we use the knowledge of how subsystems are entangled for the purpose of optimization. We gather this additional information from the Hoare triples of all invoked subroutines.

Because our aim is to optimize circuits and not to prove correctness of the entire quantum program (which includes measurements and feedback), we focus on pure states rather than employing the more general *quantum Hoare logic* [39]. In particular, we introduce a formalism to describe the entanglement between the qubits of the system throughout the execution of the quantum circuit. This entails statements that assert entanglement descriptions (to be defined next), that is, statements of the form

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

where q,r refer to quantum registers and f is a function of two registers returning one register of bits. Since q,r refer to quantum registers, they may be in superposition and entangled with other qubits in the system. As a result, we must assign 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 5.1. Entanglement description assertion. An *entanglement description assertion* on the n-qubit quantum state $|\psi\rangle$ asserts $A(q,r)$ on $|\psi\rangle$, where

$$A(q,r) = \mathsf{q}\ \text{cmp}\ f(\mathsf{q},\ \mathsf{r})\ ,$$

with cmp being 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 q,r refering to quantum registers consisting of k and m qubits, respectively. Asserting $A(q,r)$ on $|\psi\rangle$ means that

$$\forall i \in \{0,\ldots,2^n-1\} : (|\alpha_i| > 0 \implies A(\mathbf{q}(i), \mathbf{r}(i)))\ ,$$

where $\mathbf{q}(i), \mathbf{r}(i)$ extract the bits corresponding to the quantum registers q and r, respectively, from the computational basis state $|i\rangle = |i_{n-1},\ldots,i_0\rangle$.

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 5.2. To express that a qubit c is in a definite state $|0\rangle$, let $f(\cdot,\cdot) = 0$ and cmp be the equals comparison operator in the above definition. Then $A(c,\cdot) = (\mathsf{c} == 0)$. 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 the controlled gate 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 5.3. Triviality condition. A *triviality condition* of a quantum operation U is specified using an *entanglement description assertion* that asserts $A(q,r)$ on the quantum state $|\psi\rangle$. The following holds

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

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

Example 5.4. 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 can 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 in terms of potential for optimization, consider the following example.

Example 5.5. 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 $\{q_0 == q_1\}$. In particular, given that the preconditions are satisfied, the Bell-pair preparation circuit in Fig. 1 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

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

holds, 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.

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. While the Bell-pair Swap example may be a rather contrived one, we will now present more involved examples which are of practical interest and which enable significant resource savings.

6 Practical example: Optimizing floating-point arithmetic

In this section, we discuss the optimization of floating-point arithmetic using entanglement description assertions.

Besides functions that are inherently quantum such as the quantum Fourier transform or the Hadamard transform [29], quantum computers also need to evaluate classical functions albeit 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.

Examples where the evaluation of such classical functions on a quantum computer is necessary are 1) Shor's algorithm for factoring integers, which requires an implementation of modular exponentiation [32], 2) HHL algorithm for solving linear systems of equations, which requires computing the reciprocal [16], and 3) certain algorithms for solving quantum chemistry problems. Babbush et al. [2] 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 provide such functionality, one may start with implementing basic modules for floating-point arithmetic such as addition and multiplication [12]. These modules can then be combined to enable evaluating polynomials and further higher-level mathematical functions.

As a practical example for our optimization methodology, we consider a subroutine which 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. 2 and a circuit which shifts the mantissa $|x\rangle$ by $|p\rangle$ positions is depicted in Fig. 3. 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 $|p\rangle$. 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.

However, identifying this optimization opportunity in the complete circuit (combine circuits in Fig. 2 and Fig. 3) is non-trivial 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

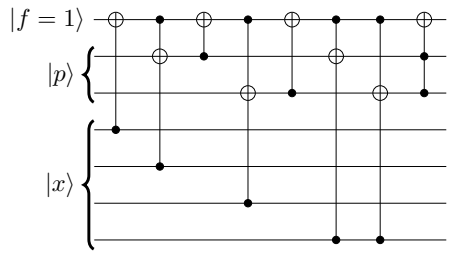


Figure 2. Example of a circuit which finds the first nonzero bit of $|x\rangle$ and stores its position in $|p\rangle$ where $|x\rangle$ is a 4-qubit register and the position register $|p\rangle$ consists of two qubits [12]. The flag qubit $|f\rangle$ is one as long as the first one has not been found.

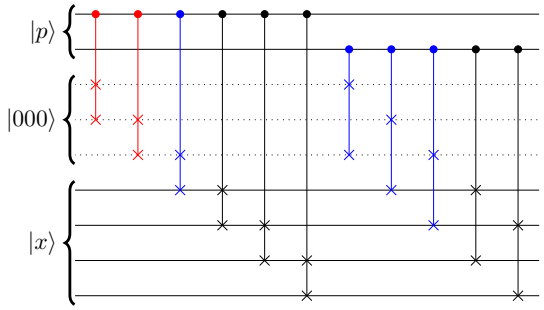


Figure 3. Optimization of the shift circuit (from [12]) which can be performed if $|p\rangle$ contains the position of the first nonzero bit. Red Fredkin gates can be removed via regular constant-folding. Blue Fredkin gates can be removed using the postconditions of the gates in Fig. 2 on $|p\rangle$. As a result, all $2^{n_p} - 1$ work qubits can be eliminated (dotted lines).

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. 3, which achieves the desired shift. Clearly, the red Fredkin gates can be removed since they act on newly allocated qubits which are zero (the postcondition of `q = allocate(n)` is that $q == 0$). 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. 3 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.

While in this example we used explicit postconditions of the subroutine which determines the position of the first nonzero bit, we demonstrate in the implementation section that it is enough to provide post- and triviality conditions of three operations—NOT, Swap, and Allocate—, in addition to the triviality condition of the control modifier, which turns a given gate U into its controlled version U^c.

Furthermore, we note that such optimization opportunities also arise when using a fixed-point representation. For example, carrying out range reductions by 2^k in order to evaluate functions such as

$$\log_2 y = \log_2(x2^{-k}) = \log_2(x) + \log_2(2^{-k}) = \log_2(x) - k \,,$$

with $x \in [1,2)$ allows for a very similar optimization opportunity: When determining x and k, one can shift y without using ancilla qubits, which is analogous to our optimization for floating-point renormalization.

7 Formalization and generalization

In this section, we formalize the deduction rules necessary to carry out all optimization examples mentioned thus far before introducing a generalization which is strictly more powerful.

7.1 Formalization of our basic methodology

In order to formalize the basics of our methodology, we first define the Hoare triples of the quantum subroutines that are required for our examples.

$$\begin{gathered} \left\{ q = \emptyset; n \in \mathbb{N} \right\} q = Alloc(n) \left\{ q = |0\rangle^{\otimes n} \right\} \\ \left\{ q = |0\rangle^{\otimes n} \right\} Dealloc(q) \left\{ q = \emptyset \right\} \\ \left\{ q_i = A, q_j = B \right\} Swap(q_i, q_j) \left\{ q_i = B, q_j = A \right\} \\ \left\{ q = A, A \in \{0,1\} \right\} X(q) \left\{ q = A \oplus 1 \right\} \end{gathered}$$

where $X(q)$ denotes application of a Pauli-X [3] 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 which we already used in our examples.

In addition to the Hoare triples 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 corresponding Hoare triple is

$$\left\{ c \in \{0,1\}, q = |\psi\rangle \right\} control(U)(c,q) \left\{ q = U^c |\psi\rangle \right\} \,,$$

where U^c denotes U raised to the c-th power, i.e., it is U if $c = 1$ and $U^c = \mathbb{1}$ if $c = 0$.

The Hoare triple of the control modifier can be combined in arbitrary ways with the Hoare triples of our subroutines. In particular, a combination of the Swap routine with the control modifier yields the rules that were used to remove trivial Fredkin gates in the circuit in Fig. 3. Combining it with the NOT or Pauli X gate, on the other hand, lets us optimize the Bell-pair example where, after an initial Hadamard gate H [3] on $|00\rangle$, the controlled NOT gate was applied as follows

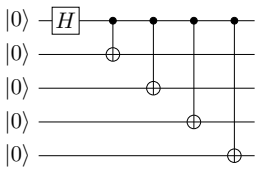
$$H_1 |0\rangle |0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) |0\rangle \overset{CNOT}{\mapsto} \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \,.$$

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

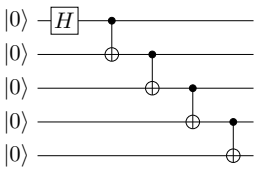
$$\{q_1 = X^{q_0} |0\rangle\} \implies \{q_1 == q_0\} \,,$$

by combining the two Hoare triples. Together with the triviality condition of the Swap gate acting on two qubits q_i and q_j,

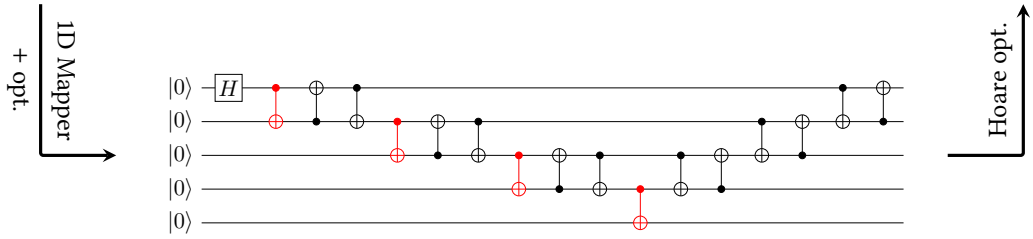
$$\{q_i == q_j\} \,,$$



(a) Original circuit for entangling all qubits



(c) Circuit for LNN after optimization.



(b) Circuit for LNN before Hoare optimization.

Figure 4. Optimizing a chain of CNOTs for a linear nearest-neighbor (LNN) architecture such as the 9-qubit chip by Google [22] by employing the Hoare triple for CNOT gates. The benefit of our optimization can be seen clearly when comparing the circuits in **(b)** and **(c)**: No extra CNOTs due to Swaps [25] are necessary in **(c)**, resulting in much lower gate count and circuit depth.

we can again remove the Swap gate from the circuit of the Bell-pair example.

Similarly, the Hoare triple for CNOT can be used to identify the optimization opportunity in the following example.

Example 7.1. 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. 4. 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 Hoare logic 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. 4**(c)**.

7.2 Generalized optimization methodology

Generalizing the basic methodology above allows to greatly increase its optimization capabilities. Thus 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 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 7.2. 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) $SUS^\dagger = |0\cdots0\rangle\langle0\cdots0| \otimes \mathbb{1} + |1\cdots1\rangle\langle1\cdots1| \otimes U'$,

(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 $\mathcal{S}$, and (3) $\mathcal{S}$ is the largest such set.

For instructions where U is not unitary, the set of control qubits is empty.

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

Example 7.3. As an example of an instruction where multiple choices exist for the set of control qubits, consider the Z^c operation applied to $|q_1q_0\rangle$, where Z acts with a (-1)–phase on $|1\rangle$ and leaves $|0\rangle$ invariant. It is easy to check that

$$\begin{aligned} Z^c &= |0\rangle\langle0| \otimes \mathbb{1} + |1\rangle\langle1| \otimes Z \\ &= \mathbb{1} \otimes |0\rangle\langle0| + Z \otimes |1\rangle\langle1| \ , \end{aligned}$$

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 nonunique.

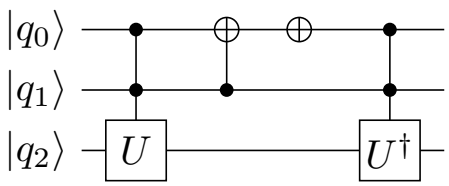


Figure 5. Simple example of our multi-gate optimization methodology: The first gate is applied if and only if the last gate is applied. Since the U gate is the inverse of $U^\dagger$, we can cancel the two doubly-controlled gates.

Definition 7.4. Target qubit. A qubit q in an instruction $U|\ldots, q, \ldots\rangle$ is called a *target qubit* if it is not in the set of control qubits of the instruction $U|\ldots, q, \ldots\rangle$.

Definition 7.5. 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 $U_1, \ldots, U_M$ denote the t-qubit gate matrices of these instructions. An optimization can be performed if

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

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. 5, where the two doubly-controlled gates can be canceled using the reasoning above.

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

Example 7.6. Consider a circuit which performs addition modulo a number N that is 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 circuit. When using the addition circuit by Takahashi et al. [37], the optimizer needs to remove the two red multi-controlled NOT gates in Fig. 6 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 Sec. 9, which discusses practical results.

8 Implementation using ProjectQ and Z3

In this section, we discuss our implementation of this optimization methodology. For each quantum operation for which we would like to add nontrivial optimization capabilities using our approach, we require

1. Postconditions
2. Triviality conditions

Additionally, preconditions may be supplied which would allow to test the program for correctness. 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 $U_1, \ldots, U_M$ acts as the identity. The latter is already available in ProjectQ.

We extend the definitions of several gate operations in ProjectQ with their respective post- and triviality conditions by providing additional member functions which employ the Z3 Theorem Prover package [7] to express these conditions. These member functions are invoked by our custom optimizing compiler engine, which then employs 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 length of quantum registers is determined since all classical inputs to the quantum program have been supplied. The circuit can thus be optimized specifically to the problem size in question—a feature that is crucial especially for near- and intermediate-term devices which have very limited resources, making such additional optimizations very valuable. In turn, 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 [7] due to the difficulty of, e.g., constructing appropriate induction rules [6]. Since all classical parameters have a definite value upon circuit generation, we can unroll many quantified statements and thereby generate statements that are much easier to (dis)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)
```

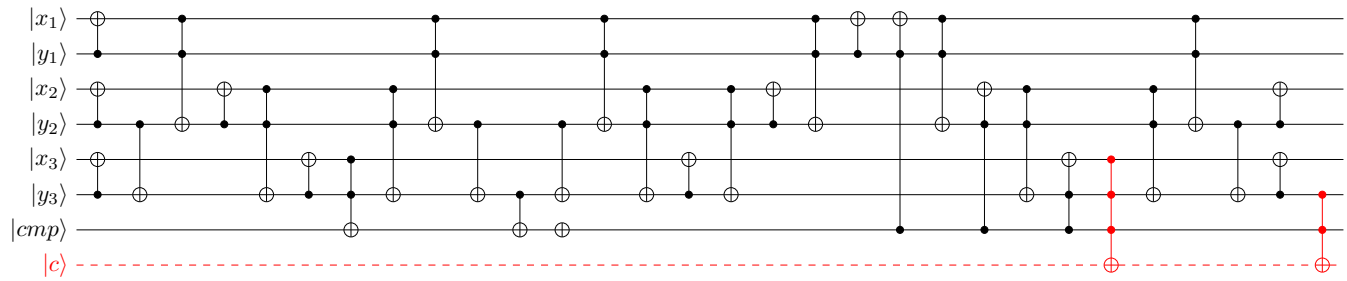


Figure 6. Three-qubit example of a modular adder subroutine which performs the modular reduction. It consists of a comparison, the result of which is stored in the qubit $|cmp\rangle$, and a conditional subtraction. In this setting, our generalized methodology is able to deduce that the two red multi-controlled 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 [5].

```
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[cqb1], v[cqb2], ...)`, which is true if and only if all variables `v[cqbi]` corresponding to control qubits `cqbi` 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_op()
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 which 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.

Also, 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 which agrees with all previous postconditions and which 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.

9 Results and comparison

In this section, we report the results that were obtained using the implementation of our optimization methodology.

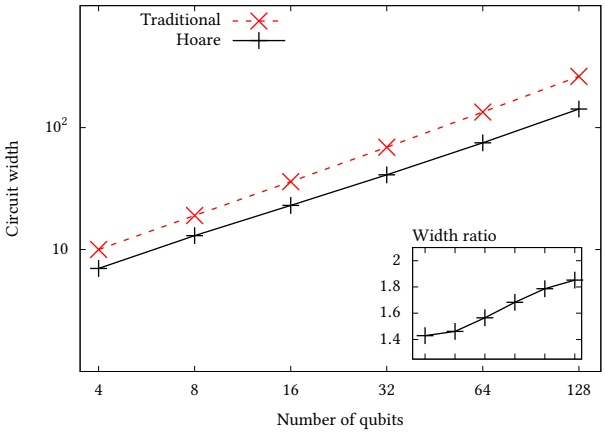


Figure 7. Optimizer comparison for the floating-point renormalization circuit with different number of qubits n for the size of the mantissa. The position register was chosen as $p := \lceil \log_2 n \rceil$. Compared to the state of the art, the Hoare optimizer achieves a reduction in circuit width between 1.4× and 1.9×. The circuit area (width × depth) is reduced by approx. 2× for all sizes.

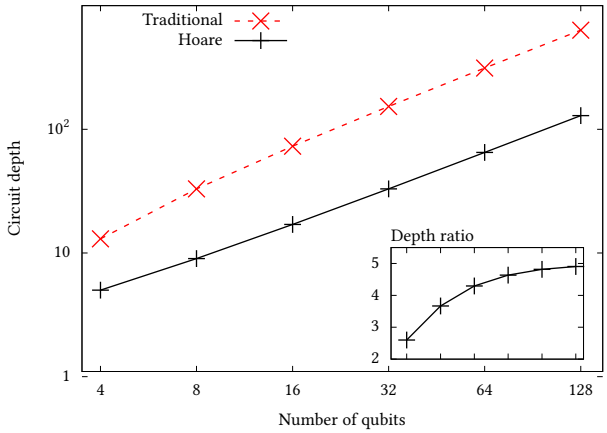


Figure 8. Optimizer comparison for the entangling circuit on n qubits depicted in Fig. 4. The Hoare optimizer achieves a 5× improvement in circuit depth for large n.

Number of qubits n	Max. circuit width	DAG depth
4	9 (11)	168 (315)
8	17 (19)	592 (639)
16	33 (35)	1240 (1287)
32	65 (67)	2536 (2583)
64	129 (131)	5128 (5175)
128	257 (259)	10312 (10359)

Table 1. Optimizer comparison for a modular reduction circuit on n qubits using the addition circuit by Takahashi et al. [37]. Numbers in parentheses correspond to results from state-of-the-art optimizers. We note that this optimization is only possible using our multi-gate or *generalized* approach.

We analyze the performance of our Hoare logic based optimizer with respect to different quantum circuits. The first circuit performs floating-point mantissa renormalization, see Figs. 2 and 3, the second entangles a linear chain of qubits, see Fig. 4, and the third performs modular reduction, see Fig. 6, which is a subroutine that is used in constructing a modular adder. For all circuits, we compare two compiler setups—one which features a simple local optimizer capable of merging/canceling subsequent operations that act on the same qubits, and a second configuration which additionally contains our Hoare logic based optimizer. As a gate set, we choose $\{\text{CNOT}, \text{X}, \text{H}, \text{S}, T, T^\dagger\}$ for all configurations.

In order to compare these compiler 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. By *depth* we mean the depth of the directed acyclic graph (DAG) associated with the circuit.

The comparisons can be found in Fig. 7 and Fig. 8 for the first and second circuit, respectively. Both cases clearly demonstrate the benefits of our Hoare logic based optimizer, which is able to reduce the circuit area (width × depth) by a factor of approximately 2× and 5× for the first and second circuit, respectively.

In the first circuit, our optimizer is able to eliminate $2^{n_P} - 1$ ancilla qubits in addition to a few Fredkin gates. It is thus to be expected that the circuit width is reduced by approximately a factor of two. Due to the cancellation of Fredkin gates, the depth is also slightly reduced.

In the second circuit, all CNOT gates resulting from swap operations can be removed when using the Hoare based optimization strategy. We thus expect the circuit depth to grow by $4(n-2)$ gates for $n \geq 2$ when turning off Hoare logic optimization. The ratio between the resulting circuit depths for $n \geq 2$ is thus

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

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

The third circuit, which is a subroutine for modular addition, can be optimized by identifying a pattern similar (but more complex) to the one shown in Fig. 5. 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,

this additional qubit can be removed from the circuit. Furthermore, due to the removal of multi-controlled NOT gates, no extra work qubits are required for Toffoli ladders [3].

10 Summary and future work

We have presented an optimization methodology that extends the scope of automatic circuit optimizations. In particular, our methodology allows to carry out certain optimizations that are typically performed by humans. This is achieved by taking into account post- and triviality conditions of all subroutines that get invoked by the quantum program that is being optimized. Our implementation in ProjectQ has managed to achieve up to a 5× reduction in circuit area for our examples when compared to the state of the art.

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 [14]: 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 [14]. Automating such procedures would thus allow for the same benefits without the need for labor-intensive manual code optimization.

Acknowledgments

This work was supported by Microsoft and by the Swiss National Science Foundation through the National Competence Center for Research NCCR QSIT.

References

[1] 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.

[2] 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.

[3] 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).

[4] Stephane Beauregard. 2002. Circuit for Shor's algorithm using 2n+ 3 qubits. *arXiv preprint quant-ph/0205095* (2002).

[5] CH Bennett. 1973. Logical reversibility of computation. *MaxwellâĂŹs Demon. Entropy, Information, Computing* (1973), 197–204.

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

[7] 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.

[8] 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

[9] 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

[10] 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.

[11] 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.

[12] Thomas Haener, Mathias Soeken, Martin Roetteler, and Krysta M Svore. 2018. Quantum circuits for floating-point arithmetic. In *International Conference on Reversible Computation.* Springer, 162–174.

[13] 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

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

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

[16] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. 2009. Quantum algorithm for linear systems of equations. *Physical review letters* 103, 15 (2009), 150502.

[17] 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

[18] Charles Antony Richard Hoare. 1969. An axiomatic basis for computer programming. *Commun. ACM* 12, 10 (1969), 576–580.

[19] IBM. 2018. QISKit. (2018). https://qiskit.org

[20] 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

[21] 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. http://stacks.iop.org/1367-2630/14/i=11/a=115023

[22] 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.

[23] 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

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

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

[26] Jarrod R. McClean, Ryan Babbush, Peter J. Love, and AlÃąn Aspuru-Guzik. 2014. Exploiting Locality in Quantum Computation for Quantum Chemistry. *The Journal of Physical Chemistry Letters* 5, 24 (2014), 4368–4380. https://doi.org/10.1021/jz501649m arXiv:https://doi.org/10.1021/jz501649m PMID: 26273989.

[27] 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.

[28] 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.

[29] Michael A Nielsen and Isaac Chuang. 2010. Quantum computation and quantum information. (2010).

[30] David Poulin, Matthew B. Hastings, Dave Wecker, Nathan Wiebe, Andrew C. Doberty, and Matthias Troyer. 2015. The Trotter Step Size Required for Accurate Quantum Simulation of Quantum Chemistry. *Quantum Info. Comput.* 15, 5-6 (April 2015), 361–384. http://dl.acm.org/citation.cfm?id=2871401.2871402

[31] 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* (2017), 201619152.

[32] 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

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

[34] 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

[35] 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 (RWDSL2018)*. ACM, New York, NY, USA, Article 7, 10 pages. https://doi.org/10.1145/3183895.3183901

[36] Yasuhiro Takahashi and Noboru Kunihiro. 2006. A Quantum Circuit for Shor's Factoring Algorithm Using 2N + 2 Qubits. *Quantum Info. Comput.* 6, 2 (March 2006), 184–192. http://dl.acm.org/citation.cfm?id=2011665.2011669

[37] 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

[38] Dave Wecker, Bela Bauer, Bryan K. Clark, Matthew B. Hastings, and Matthias Troyer. 2014. Gate-count estimates for performing quantum chemistry on small quantum computers. *Phys. Rev. A* 90 (Aug 2014), 022305. Issue 2. https://doi.org/10.1103/PhysRevA.90.022305

[39] 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