

Compiling Quantum Computations into Elementary Operations

Krysta M. Svore

Dept. of Computer Science, Columbia University, New York, NY 10027

kmsvore@cs.columbia.edu

I pledge my honor that this is a representation of my work as a new investigator.

Abstract

Shor’s polynomial time algorithm for factoring numbers on a quantum computer has considerably escalated interest in quantum computation. However, the current inability to build a scalable quantum computer prevents the usage of such an algorithm in any computationally significant setting. It is necessary to optimize both the size and speed of a quantum algorithm to decrease the number of qubits and the coherence time needed to implement the algorithm on a quantum computer. For this reason, the design and optimization of quantum circuits is central to quantum computation. This paper presents a machine-independent compilation framework based on matrix decomposition for reducing quantum computations into elementary unitary operations for any future quantum computer technology. We review the decomposition techniques of Aho and Svore, Knill, and Vartiainen et al. and the corresponding bounds on the number of elementary gates in the resulting circuit.

1. Introduction

In the last decade, with the invention of Shor’s algorithm for prime factorization [11], Grover’s database search algorithm [8] and other important quantum algorithms that run more efficiently than the corresponding known classical algorithms, interest in quantum computing has grown. While mathematicians and computer scientists have concentrated on the development of algorithms that exploit the “magic” of quantum computation, such as superposition and quantum entanglement, physicists have been developing physical technologies to execute the algorithms. Any practical technology will need to provide a sufficient number of qubits and a suitable coherence time to produce scalable, reliable computations.

The primary model for mapping a quantum algorithm to a quantum technology is the quantum circuit model [6]. The quantum circuit model is analogous to the classical circuit model of bits, wires, and gates, and consists of quantum bits, quantum wires, and quantum gates, where quantum wires provide communication between the sequential quantum gates by connecting one computation to the next. This model provides a framework in which to consider the optimization of quantum computations and their corresponding

quantum circuits. An arbitrary n -qubit unitary gate has 4^n degrees of freedom and may not be able to be decomposed into an efficient circuit. However, certain classes of gates can be expressed with a polynomial-number of elementary gates, decreasing the required coherence time of the circuit. It is also important for any quantum computation to use as few elementary unitary operations as possible in the early phases of compilation since the later phases of quantum circuit compilation can increase the circuit size for each additional gate in the initial circuit representation [9].

In this paper, we present a framework for compiling a quantum computation into a circuit of elementary gates. The compilation process decomposes an arbitrary $2^n \times 2^n$ unitary matrix into a sequence of matrices which can be compiled into a circuit consisting of elementary quantum gates. This framework for quantum compilation is useful in the first phase of any general procedure for decomposing a quantum computation into an efficient quantum circuit of elementary unitary operations. In the remaining sections of this paper, we first review the necessary background and notation of quantum computation. In Section 3, we describe our framework for quantum circuit compilation. We review the most recent decomposition and circuit construction processes in Sections 4 and 5. Finally, we conclude in Section 6 with future research directions.

2. Background

Throughout this article, we use the standard *Dirac* notation for quantum states, where a quantum state ψ is written in *ket* form as $|\psi\rangle$. A *quantum bit*, or *qubit*, has a state $|0\rangle$, $|1\rangle$, or a *linear combination* of these states, called a *superposition*, written as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex numbers and $|\alpha|^2 + |\beta|^2 = 1$. A *quantum register* consists of n qubits which lie in a 2^n -dimensional complex Hilbert space. A n -qubit state can be described by the vector

$$|\psi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$$

where the *computational basis states* are of the form $|x_{n-1} \dots x_1 x_0\rangle$ and the probability of measuring state $|x\rangle$ is $|\alpha_x|^2$, where $x = x_{n-1} \dots x_1 x_0$ and α_x is a complex number. A fascinating property called *entanglement*, that can lead to quantum algorithmic speed-ups, occurs when a

state $|\psi\rangle$ cannot be written as the product of single qubit states.

In the quantum circuit model [6], a *quantum gate* on n qubits is a $2^n \times 2^n$ unitary matrix U . A *quantum circuit* consists of a composition of quantum gates $G_k \dots G_1$. Two quantum circuits are *equivalent* if the composition of their respective gates represents the same unitary matrix. To identify the matrix elements of particular quantum gates, we order our states lexicographically. In our circuit diagrams, control flows from left to right, but the order of operators in a matrix sequence is applied to the state from right to left.

Several standard quantum gates are used to build our quantum circuits. A single-qubit gate is applied to one qubit and is described by a 2×2 matrix. The most important single qubit gates are the *Pauli operators* given by

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

The X operator is similar to the classical *NOT* operation and takes the state $|x\rangle \rightarrow |x \oplus 1\rangle$, where \oplus denotes modulo 2 addition.

Multiple qubit operations are applied to several qubits and include *control gates* which perform the target operation only if the control qubits are set appropriately. The $(n-1)$ -controlled gate, written as $\Lambda_{n-1}(S)$, denotes $n-1$ qubits controlling the application of the operator S to the target qubit and is defined by

$$\Lambda_{n-1}(S)|x\rangle|\psi\rangle = |x\rangle S^{x_{n-1} \wedge \dots \wedge x_1 \wedge x_0} |\psi\rangle$$

where S represents a single qubit gate, $x = x_{n-1} \dots x_1 x_0$ and $x_{n-1} \wedge \dots \wedge x_1 \wedge x_0$ in the exponent of S denotes the Boolean product of the bits x_{n-1}, \dots, x_1, x_0 . If the product of these bits is 0, then the operator is not applied.

The $\Lambda_1(X)$ gate is known as the controlled-NOT gate (*CNOT*) and performs the operation $|x, y\rangle \rightarrow |x, x \oplus y\rangle$. In matrix form, $\Lambda_1(X)$ is

$$\Lambda_1(X) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (1)$$

Circuits are often designed using a specific set of gates according to a given technology. A set of quantum gates is *exactly universal* if it can represent any unitary operation exactly by a composition of its gates; a set is *approximately universal* if it can approximate any unitary operation to an arbitrary accuracy with a composition of its gates [5]. Since there are noncountably many operations, exact universality requires an infinite generating set of quantum gates, while approximate universality can be achieved by certain discrete sets of gates. In this paper, we consider exact universality.

3. A Framework for Quantum Circuit Compilation

We now describe the front end of our machine-independent quantum circuit compiler framework that generates for an arbitrary unitary matrix U a circuit of elementary gates, i.e., $\Lambda_1(X)$ and single-qubit gates. The compilation steps of this framework are shown in Figure 1.

This first phase, *operator decomposition*, takes as input a $2^n \times 2^n$ unitary matrix U and an *ordering of decomposition* and outputs a sequence of matrices $V_1 \dots V_k$ such that $V_1 \dots V_k = U$, where the value of k depends on the chosen decomposition algorithm. This first step breaks the general unitary matrix U into smaller components that allow for simpler circuit construction. In a quantum circuit, we would like simple, easy-to-apply gates that are fault-tolerant and reliable. General multiple qubit unitary operations and $\Lambda_{n-1}(S)$ gates are difficult to implement in practice, while it is assumed elementary operations can easily be implemented.

The $V_1 \dots V_k$ output is then converted into an optimized intermediate circuit, $G_m \dots G_1$, of intermediate gates. We require that the intermediate gate choice be universal, since an arbitrary unitary matrix is given as input. A possible intermediate gate choice is the set of *two-level* gates, where *two-level* indicates the gate acts nontrivially on two or fewer basis vectors. An optional optimization can be performed on the circuit using circuit identities as rewriting rules, or similar optimization methods, that results in a more efficient sequence of gates.

After intermediate circuit construction, the output $G_m \dots G_1$ is decomposed into a circuit of chosen basis gates using standard techniques of Barenco et al. [3]. This process decomposes a $\Lambda_{n-1}(S)$ gate, where S is an arbitrary single qubit gate, into $O(n^2)$ elementary gates. If the determinant of S equals 1, this decomposition requires only $O(n)$ elementary operations. This circuit is then optimized over some objective function, such as the depth or size of the circuit.

In this paper, we focus on a machine-independent framework, although the choice of basis gates should be made with knowledge of the desired technology. A corresponding machine-dependent model of compilation must also be developed for each quantum technology. In the next sections, we describe the different decomposition algorithms, circuit construction processes, and intermediate gate choices and compare them to the conventional decomposition process given in [10, 3] that achieves an upper bound of $O(n^3 4^n)$ elementary gates.

4. Two-Level Decomposition

In this section, we describe the two-level decomposition algorithm, its different settings, and the corresponding intermediate circuit constructions and optimizations. We present three possible orderings of decomposition, two possible cir-

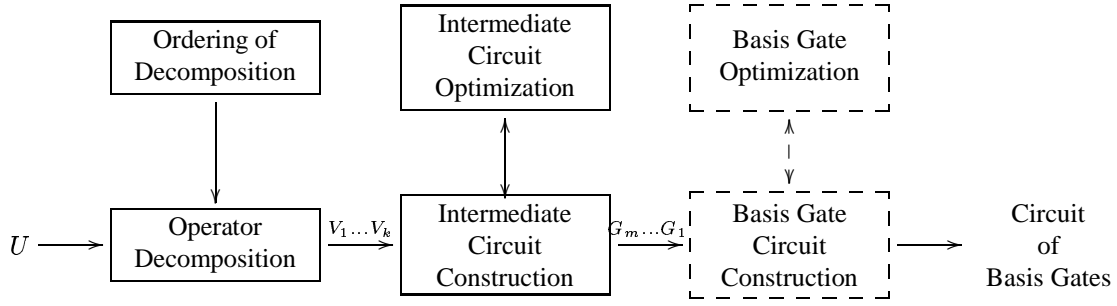


Figure 1: The compilation steps of exact quantum circuit generation.

circuit constructions, and two intermediate circuit optimizations that greatly reduce the number of gates in the conventional decomposition process. The ordering of decomposition has a significant effect on the size of the resulting circuit as the results in the following sections indicate.

Two-level decomposition decomposes an arbitrary unitary matrix $U = \{u_{r,c}\}_{r,c=0}^{2^n-1}$ into a composition of two-level matrices $V_1 \dots V_k$ such that $V_1 \dots V_k = U$, or equivalently $V_k^\dagger \dots V_1^\dagger U = I$, where \dagger indicates the adjoint operation. We can easily verify that $k \leq 2^{n-1}(2^n - 1)$. In linear algebra, this process is called *QR-Factorization with quantum Givens operations* [4].

Each V_j , $j \in \{1, 2, \dots, k\}$, nontrivially acts on the basis vectors $|e_r\rangle$ and $|e_c\rangle$ and thus nullifies the $u_{r,c}$ element of U . We call this pair of basis vectors the *ordering pair* and denote it as (r, c) . We will use the notation $V_{r,c}$ when explicitly indicating the ordering pair of V_j , leaving off the subscript j for simplicity. The matrix representation of $V_{r,c}$ matches the identity matrix except for entries $v_{c,c}, v_{c,r}, v_{r,c}, v_{r,r}$. These four entries form the *component matrix* of $V_{r,c}$, written $\tilde{V}_{r,c}$, and are given by

$$\tilde{V}_{r,c} = \begin{bmatrix} v_{c,c} & v_{c,r} \\ v_{r,c} & v_{r,r} \end{bmatrix} = \frac{1}{\sqrt{|u_{c,c}|^2 + |u_{r,c}|^2}} \begin{bmatrix} u_{c,c}^* & u_{r,c}^* \\ u_{r,c} & -u_{c,c} \end{bmatrix}$$

where $*$ indicates the complex conjugate operation.

We define the *order of two-level decomposition* as the sequence of ordering pairs that are nontrivially acted on by the two-level matrices in the decomposition $V_1 \dots V_k$. This is equivalently the selection of the basis vectors and their ordering in the matrix representation. The sequence of ordering pairs defines the order of two-level decomposition. To avoid repetition in a two-level decomposition, we only allow ordering pairs (r, c) , where $r > c$. The first number of an ordering pair represents a row and the second a column in the matrix representation.

We can consider two types of orderings: *fixed-column* and *non-fixed-column*. In a *fixed-column* ordering, we choose a fixed computational basis order. That is, in terms of the matrix representation, if in all our sequences of ordering pairs, we begin with the pairs for column 0 followed by those for 1, followed by those for column 2, and so on

up to column $2^n - 2$, then this is a *fixed-column* ordering; otherwise, it is a *non-fixed-column* ordering. The conventional ordering for two-level decomposition is a fixed-column ordering since it orders the binary representation of the rows and columns lexicographically, and has the pairs $(c+1, c), (c+2, c), \dots, (2^n-1, c)$ for column c followed by the pairs $(c+2, c+1), (c+3, c+1), \dots, (2^n-1, c+1)$ for column $c+1$, and so on. Later, we will describe two decomposition algorithms using a fixed-column ordering [2, 12] and one in which the ordering is not fixed [13].

Two-level decomposition iteratively applies the matrices V_j to perform the nullification process. In the case of fixed-column ordering, it iteratively applies this process to the $2^n \times 2^n$ matrix U , then to the $2^n - 1 \times 2^n - 1$ unitary submatrix in the lower right-hand corner of $V_1^\dagger U$, ultimately decomposing U into a product of two-level unitary matrices. It continues in this fashion until column 0 has a 1 in the top entry and 0's everywhere else. The fixed-column decomposition is given by

$$\left(\prod_{r,r>c} \prod_{c=0}^{2^n-2} V_{r,c}^\dagger \right) U = I$$

where the order of r is determined by the *ordering of decomposition*. For a more detailed description of this process, refer to [10, 2].

In the *intermediate circuit construction* step, we can use different methods to construct the intermediate circuit for the sequence $V_1 \dots V_k$. In *multi-control circuit construction*, we construct a circuit of $\Lambda_{n-1}(\tilde{V}_j)$ and $\Lambda_{n-1}(X)$ gates, where \tilde{V}_j is the component matrix. The idea is to effect a state change from $|e_c\rangle$ to $|e_r\rangle$, apply the $\Lambda_{n-1}(\tilde{V}_j)$ gate, and then return the state to the state $|e_c\rangle$. The state change sequences consist of a sequence of swaps and are thus implemented using $\Lambda_{n-1}(X)$ gates. For details, refer to [10, 2].

An alternate *intermediate circuit construction* step, called *single-control circuit construction*, compiles $V_1 \dots V_k$ into a circuit of $\Lambda_{n-1}(\tilde{V}_j)$, X , and $\Lambda_1(X)$ gates in the manner of [4]. The circuit must act only on the two states $|e_c\rangle$ and $|e_r\rangle$. We can construct the circuit easily as follows. We know the two vectors differ in at least one

bit position, say the k^{th} bit. If other additional bits differ, then we place X on that qubit, controlled by the k^{th} qubit. Then, we apply X on every pair of qubits that have state $|0\rangle$. Note that the $\Lambda_1(X)$ gate may have effected a qubit to be $|0\rangle$, so this pair also requires a X gate. Finally, apply $\Lambda_{n-1}(\tilde{V}_j)$, with the target on the k^{th} qubit and then reverse the entire circuit.

The conventional algorithm uses multi-control circuit construction with no intermediate optimization step and uses fixed-column two-level decomposition with the conventional ordering given above. This results in a circuit of $O(n^3 4^n)$ elementary gates. In the next section, we present different orderings and intermediate optimizations that improve this bound.

4.1. The Palindromic Optimization Algorithm (POA)

In this section, we review the *Palindromic Optimization Algorithm* (POA), that uses fixed-column two-level decomposition, an ordering of decomposition called the *palindrome ordering*, multi-control circuit construction, and the *palindrome transform optimization* [2]. POA improves the $O(n^3 4^n)$ size circuit of elementary operations resulting from conventional two-level decomposition to $O(n^2 4^n)$ elementary gates.

We call a gate G *self inverting* if $GG = I$. A sequence of self-inverting gates of the form $G_1 G_2 \dots G_{m-1} G_m G_m G_{m-1} \dots G_2 G_1$ simplifies to the empty sequence and is called *self-annihilating*. In the case of the $\Lambda_{n-1}(\tilde{V}_j)$ and $\Lambda_{n-1}(X)$ circuit construction process, the subsequences of gates are the form $G_1 G_2 \dots G_m \beta G_m \dots G_2 G_1$ for $m \geq 0$, where β is a $\Lambda_{n-1}(\tilde{V}_j)$ gate and the $G_i, i \in \{1, \dots, m\}$, are $\Lambda_{n-1}(X)$ gates and are self-inverting. We call a sequence of this form a *partial palindrome*.

Let $\alpha = G_1 G_2 \dots G_m$ and α^R denote $G_m \dots G_2 G_1$. Then we define the *overlap* between two partial palindromes $\alpha_1 \beta_1 \alpha_1^R, \alpha_2 \beta_2 \alpha_2^R$ to be the longest suffix γ^R of α_1^R , or equivalently the longest prefix γ of α_2 , such that $\gamma^R \gamma$ is a self-annihilating sequence.

By entering the partial palindromes into a trie [1] we call the *palindrome transform trie*, we identify the maximal length common prefixes for all partial palindromes. By printing the labels of the leaves of the trie in a depth-first-search order, we group those subsequences of partial palindromes which must be juxtaposed to achieve the maximal possible overlaps. In other words, we determine the necessary ordering, called the *palindrome ordering*, of basis vectors to produce a circuit that minimizes the number of $\Lambda_{n-1}(X)$ gates in the circuit for a fixed-column two-level decomposition and multi-control circuit construction. An enumerative method of determining the palindrome ordering without using the trie data structure is given in [2].

The intermediate optimization, called the *palindrome transform optimization*, uses knowledge of partial palin-

dromes and of the ordering of decomposition to annihilate self-inverting gates. An example circuit resulting from the conventional algorithm for an arbitrary $2^n \times 2^n$ matrix U is given in Figure 2. The circuit resulting from using POA is given in Figure 3. The following table gives the number of $\Lambda_{n-1}(\tilde{V}_j)$ and $\Lambda_{n-1}(X)$ gates in an n -qubit circuit using POA and the conventional algorithm.

n	POA	Conventional
2	8	10
3	50	68
4	246	392
5	1086	2064
6	4558	10272
7	18670	49216

4.2. The Gray Code Optimization Algorithm (GCA)

In this section, we review the *Gray Code Optimization Algorithm* (GCA) given in [12]. It uses fixed-column two-level decomposition, an ordering of decomposition called the *graycode ordering*, single-control circuit construction [4], and the *palindrome transform optimization* to generate a circuit of $O(n 4^n)$ elementary gates.

Recall that in single-control construction, we must choose a k to act as the target qubit. We would like our choice of k , in conjunction with our special ordering, to result in a smaller circuit. If we choose k from the left, i.e., we check from the highest order bit to the lowest order bit for the first differing bit and label it k , then we achieve a significant reduction in circuit size. If we choose another order, then we may not achieve as dramatic of a reduction.

Automatically, this circuit construction process, with the conventional ordering of decomposition, results in a circuit of $O(n 4^n)$ elementary operations. We can further improve the dominating constant by choosing an ordering of decomposition based on graycodes as described in [12]. The graycode ordering orders the V_j such that the single-control circuit construction creates partial palindromes whose overlaps can be annihilated. The *palindrome transform optimization* then annihilates the self-inverting gates in the intermediate circuit.

The following table gives the number of $\Lambda_{n-1}(\tilde{V}_j)$, $\Lambda_1(X)$, and X gates in the circuit of a $2^n \times 2^n$ arbitrary unitary matrix resulting from applying GCA and fixed-column two-level decomposition with single-control circuit construction and the conventional ordering.

n	GCA	With Conv. Ordering
2	12	16
3	62	124
4	266	752
5	1074	4048
6	4258	20352
7	16834	97984

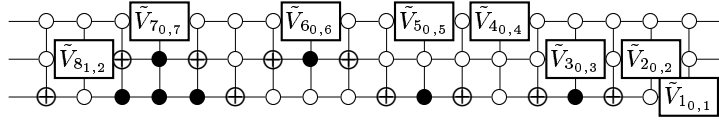


Figure 2: A subsequence of the unoptimized circuit for an arbitrary $2^3 \times 2^3$ unitary matrix using the conventional ordering.

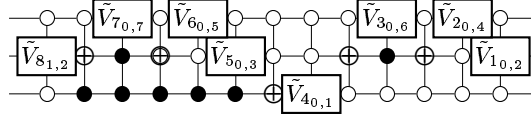


Figure 3: A subsequence of the optimized circuit for a $2^3 \times 2^3$ unitary matrix using POA.

An example of an unoptimized circuit for an arbitrary $2^n \times 2^n$ unitary matrix is given in Figure 4 and the circuit resulting from GCA is given in Figure 5.

4.3. The Gray Code Basis Optimization Algorithm (GCBA)

In this section, we present the compilation given in [13], called the *Gray Code Basis Optimization Algorithm* (GCBA), that achieves a circuit of $O(4^n)$ elementary gates. This algorithm uses non-fixed-column two-level decomposition, an ordering of decomposition called the *graycode basis ordering*, multi-control circuit construction, and a control bit removal optimization. It further generalizes the use of graycode orderings in the non-fixed-column setting of [12].

Vartiainen et al. use a transformation to perform a change of basis from the binary basis to the graycode basis of the V_j . This creates a non-fixed-column ordering of decomposition, called the graycode basis ordering, and we can write the factorization as

$$\left(\prod_{r=1}^{2^n-2} \prod_{c=0}^{2^n-2} V_{\gamma(r), \gamma(c)}^\dagger \right) U = I$$

where $\gamma(j)$ is the graycode transformation function described in [13].

By using the graycode basis ordering, they are able to eliminate the need for effecting a state transformation in the multi-qubit circuit construction process, thus requiring no $\Lambda_{n-1}(X)$ gates in the resulting circuit. However, this does not improve the bound resulting from GCA, but with additional optimizations matches the theoretical lower bound given by [9]. They present an intermediate circuit optimization algorithm we call the *Control Bit Removal Optimization* to further reduce the size of the circuit to $O(4^n)$ elementary gates.

The key idea behind the control bit removal optimization is that not all controls in the $\Lambda_{n-1}(\tilde{V}_j)$ gates are necessary to achieve an exact two-level decomposition. By removing unnecessary controls, they are able to reduce the $O(n4^n)$

size circuit to a circuit consisting of approximately 30 times more $\Lambda_1(X)$ gates than the theoretical lower bound of $O(4^n)$ elementary gates [9]. For details on this circuit optimization, refer to [13].

5. Eigenvector Decomposition

In this section, we describe a decomposition algorithm entirely different from the two-level decomposition algorithm given in Section 3. This section reviews the eigenvector decomposition of Knill [9, 7] that yields a circuit of $O(n4^n)$ elementary operations. In eigenvector decomposition, we decompose a general $2^n \times 2^n$ unitary matrix U into a sequence of diagonal representations $V_1 \dots V_k$ such that $V_1 \dots V_k = U$.

In this algorithm, we must first compute the eigenvalues $e^{i\phi_j}$ and eigenvectors $|e_j\rangle$ of U . This is called a *diagonal representation* of U and is written

$$U = \sum_{j=0}^{2^n-1} e^{i\phi_j} |e_j\rangle \langle e_j|$$

We can rewrite this in the following form, which provides an easier interpretation for the circuit construction process,

$$U = \prod_{j=0}^{2^n-1} (I + (e^{i\phi_j} - 1)|e_j\rangle \langle e_j|) = \prod_{j=1}^{2^n} V_j$$

To construct the intermediate circuit from the above decomposition, we see that each V_j multiplies eigenvector $|e_j\rangle$ with its corresponding eigenvalue. We effect the state transformation from $|e_j\rangle$ to $|00\dots 0\rangle$, apply the $\Lambda_{n-1}(P)$ gate, where P is the gate

$$P = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi_j} \end{bmatrix}$$

and then effect the state transformation from $|00\dots 0\rangle$ to $|e_j\rangle$. We can assume the determinants can be set to 1, allowing us to construct this circuit using $O(n2^n)$ elementary gates. Since we require 2^n eigenvectors, the entire circuit can be constructed using $O(n4^n)$ elementary gates.

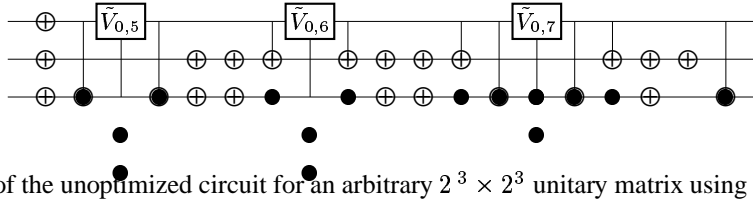


Figure 4: A subsequence of the unoptimized circuit for an arbitrary $2^3 \times 2^3$ unitary matrix using the conventional ordering.

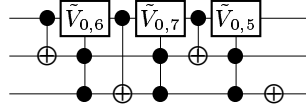


Figure 5: A subsequence of the optimized circuit for a $2^3 \times 2^3$ unitary matrix using GCA.

6. Conclusions and Future Work

We have presented a framework for compiling quantum circuits into elementary unitary operations and a powerful set of decomposition techniques for quantum circuit compilation. The following table gives a summary of the number of elementary gates in the circuits resulting from the methods described in Sections 3 and 4.

Decomposition	Method	# Elementary Gates
2-Level,Fixed	Conv.	$O(n^3 4^n)$
2-Level,Fixed	POA	$O(n^2 4^n)$
2-Level,Fixed	GCA	$O(n 4^n)$
2-Level,Non-fixed	GCBA	$O(4^n)$
Eigenvector	Eigenvector	$O(n 4^n)$

We observe that GCBA is within a constant of the theoretical lower bound, $O(4^n)$, by using a special intermediate optimization procedure. Future research should address the possibility of applying similar optimizations to GCA, eigenvector decomposition, or other decomposition techniques. In addition, further work needs to be done to determine which of these techniques is most useful in practice.

These algorithms present the significant improvements possible with optimization procedures in compilation frameworks. We hope our presentation motivates the study of quantum circuit optimization algorithms, as well as the necessary compilation and design components of a future quantum computer architecture. Currently, it is not known what classes of gates will result in polynomial-size circuits of elementary operations. Our ongoing work is focused on developing effective optimization techniques for both a machine-independent and machine-dependent quantum computer compiler, as well as important classes of quantum gates.

References

- [1] A. Aho, J. Hopcroft, and J. Ullman. Data Structures and Algorithms. Addison-Wesley, 1983.
- [2] A. Aho and K. Svore. Compiling quantum circuits using the palindrome transform. quant-ph/0311008, 2003.
- [3] A. Barenco, C. Bennett, R. Cleve, D. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Phys. Rev. A*, 52:3457-3467, 1995.
- [4] G. Cybenko. Reducing quantum computations to elementary unitary operations. *Computing in Science and Engineering*, 3(2):27-32, 2001.
- [5] D. Deutsch, A. Barenco, and A. Ekert. Universality in quantum computation. *Proc. R. Soc. London A*, 449(1937):669-677, 1995.
- [6] D. Deutsch. Quantum computational networks. *Proc. R. Soc. London A*, 425:73, 1989.
- [7] M. Grassl. Notes on implementing unitary transformations. unpublished, 2003.
- [8] L. Grover. A fast quantum mechanical algorithm for database search. *Proc. of the 28th Annual Symposium on Theory of Computing*, 1995.
- [9] E. Knill. Approximating quantum circuits. quant-ph/9905086, 1995.
- [10] M. A. Nielsen and I. L. Chuang. Quantum Computation and Quantum Information. Cambridge University Press, 2000.
- [11] P. Shor. Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. of Comput.*, 26(5):1484-1509, 1997.
- [12] K. Svore. Compiling quantum computations into elementary operations. QIP, Waterloo, Canada, 2004.
- [13] J. Vartiainen, M. Mottonen, and M. Salomaa. Efficient decomposition of quantum gates. quant-ph/0312218, 2003.