CPSC 4210 - Final Paper

R. Lowry, C. Rabl, R. Rana

Abstract

his paper explores the use of genetic programming in conjunction with a shared cube representation of reversible cascades to minimize the gate count and quantum cost of arbitrary reversible circuits. Rather than optimizing for either quantum cost or cascade length, we utilize a feature vector approach to minimize both aspects. A recent approach by Nayeem and Rice provides a foundation for the shared-cube approach to reversible logic synthesis. Due to the complexity of these operations, we expand on this foundation, combining it with a genetic algorithm to yield a solution. This approach allows us to achieve a solution more quickly than by performing a brute-force search over the problem space. We demonstrate the use of a shared cube representation from Nayeem and Rice [2011] and apply an evolutionary algorithm to find optimized representations. We consider an implementation that uses a subset of the ordering rules in Rice and Nayeem [2011] as a guideline for implementing our mutation function. We present an implementation of this approach and selected examples using the Python programming language, and compare our results to those achieved in the literature as applied to Revlib benchmarks Wille et al. [2008].

1 Unitary Matrices

Every quantum gate can be represented by a unitary transformation (in the form of a unitary matrix) whose entries are complex variables corresponding to the complex coefficients of a given particle's wave function. Unitary transformations allow us to perform actual computations with qubits since they can be realized using technologies like NMR, for instance: a qubit in an NMR machine undergoes state changes due to a changing magnetic field. These magnetic field changes are, in turn, represented by unitary matrices (Lukac et al. [2003]). Constructing useful quantum circuits in this way is analogous to early computer programmers who "programmed" massive machines like ENIAC by physically connecting relays and vacuum tubes with wires. How far we have come since then...

Definition 1. *A* **unitary matrix** *is an $n \times n$ matrix of complex coefficients which, when multiplied by its Hermitian, gives the identity matrix. Thus, for a unitary matrix U, it is true that $(U^T)^* = U^{-1}$ where $(\cdot)^*$ denotes complex conjugation.*

In addition, every 2×2 unitary matrix can be expressed using the following product (Barenco et al. [1995]). This representation is particularly useful since it corresponds to the exact rotation matrix which can be applied to the Bloch sphere representation of a qubit:

$$\begin{bmatrix} e^{i\delta} & 0 \\ 0 & e^{i\delta} \end{bmatrix} * \begin{bmatrix} e^{\frac{i\alpha}{2}} & 0 \\ 0 & e^{\frac{-i\alpha}{2}} \end{bmatrix} * \begin{bmatrix} \cos\theta/2 & \sin\theta/2 \\ -\sin\theta/2 & \cos\theta/2 \end{bmatrix} \times \begin{bmatrix} e^{i\beta/2} & 0 \\ 0 & e^{-i\beta/2} \end{bmatrix}$$

In order to create useful operations out of "quantum primitives", we can compose unitary transformations in order to come up with a permutation representation of a gate or cascade of gates. We can use the "Square-Root-of-NOT" gate to construct a NOT gate, for instance:

$$\sqrt{\text{NOT}} = \frac{1+i}{2} \begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix} \Rightarrow \sqrt{\text{NOT}} * \sqrt{\text{NOT}} = \left(\frac{1+i}{2}\right)^2 \begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix} * \begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

There are many other unitary matrices: more examples may be found in Lukac et al. [2003]. Gates such as the Pauli-X, Pauli-Y, Pauli-Z, are interesting examples to play around with.

2 Permutation Matrices

Definition 2. *A* **permutation matrix** *P is an $n \times n$ matrix created by permuting the rows of the identity matrix I_n. It is the case that $P * P^T = P^T * P = I$ and that $det(P) = 1$.*

Rather than deal with extremely large and unwieldy unitary transformations all the time, we can use permutation matrices, as described by Williams [1999]. These are a powerful tool, since an $n \times n$ permutation matrix can be used to represent a $2^n \times 2^n$ unitary operation. Additionally, since permutation matrices are sparse, they can be computed with and stored more efficiently than full matrices. As an aside, some unitary matrices are also permutation matrices (such as the NOT gate), but the gates which are "true quantum primitives", as described in Lukac et al. [2003] are only unitary.

In brief, permutation matrices encode the rows of a circuit or gate's truth table. Given the truth table for a CNOT gate, for instance, it is quite simple to construct its permutation matrix: we begin by encoding the inputs and outputs of the gate as decimal numbers, and create a mapping between them. Then, we use this mapping to construct the permutation matrix, using the following rule:

$$P = [p_{ij}] \text{ where } p_{ij} = \begin{cases} 1 & \text{if } i = n \text{ and } j = M(n) \quad \forall n \in \mathbb{Z}_k \\ 0 & \text{otherwise} \end{cases}$$

In this case, $k = 2^w$ where w is the "width" of the gate, or the number of inputs. Since CNOT has a width of 2, that means $k = 2^2 = 4$, in this case.

a	b	a'	b'
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

$\rightarrow$

n	$M(n)$
0	0
1	1
2	3
3	2

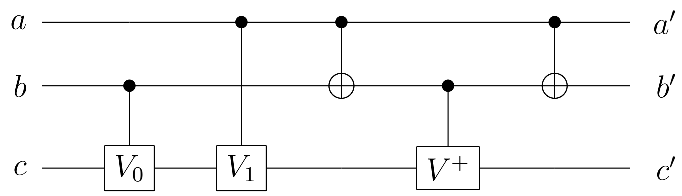
$$\rightarrow \quad P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

A useful property of permutation matrices is that they allow us to "compose" permutations. In order to do this, we use the following identity: $P_{\sigma \circ \pi} = P_\pi * P_\sigma$. Note that the order of the matrix multiplication matters, as matrices do not typically commute under multiplication. Having this composition operator makes it easy to represent cascades in a unique way. We can check that two cascades realize the same function if their output permutation matrices are identical. This provides circuit designers with an efficient way to "equate" cascades and determine which is "better".

3 Quantum Cost

Since we can represent operations on qubits using unitary transformations (which conveniently correspond to exactly one quantum operation each), we can devise a metric called "quantum cost" in order to determine whether the transformations we perform constitute an efficient synthesis of a given operation. In an NMR system, each electromagnetic pulse to which we subject a qubit has a cost: whether it is the amount of energy required to create the pulse, or the risk of the qubit decohering into a useless state (through vibrations, or other environmental perturbations), these factors may be treated as unitless "cost" variables which must be taken into account.

As quantum cost is a unitless quantity which corresponds directly to the number of unitary operations in a quantum circuit, it is a very useful metric for calculating the efficiency of an implementation of a circuit. In order to determine the quantum cost of a gate or cascade, we need to break it down into "quantum primitives" (unitary transformations). For instance, we can break down a 3-input Toffoli gate like so:



Of course, it is not immediately obvious why this construction gives us a Toffoli gate. Note that the $\sqrt{\text{NOT}}$ gates (and their Hermitian friend) do not get activated unless their control lines are 1.

So, if we pass $a = 0$ and $b = 0$ through our gate, c remains unchanged, as do a and b. If $a = 0$ and $b = 1$, then the gate that gets applied to c will be $V_0 * V^+ = I$, which is the identity, so c will be unchanged. If $a = 1$ and $b = 0$, then the gate that gets applied to c will be $V_1 * V^+ = I$, so c will be unchanged, and finally, if $a = 1$ and $b = 1$, c will be inverted because the gate that gets applied will be $V_0 * V_1 = \text{NOT}$. And thus, we have shown that a 3-input Toffoli gate may be simulated by at least five quantum primitives, and so it has a quantum cost of 5. This result is due to DiVincenzo and Smolin [1994].

4 ESOP Cube-list Representations

Any boolean function can be represented by an exclusive-or sum-of-products (ESOP) expression.This is particularly useful for reversible logic synthesis since there are existing algorithms for converting any ESOP expression into a cascade of reversible toffoli gates, thus generating a reversible circuit to implement your original function.

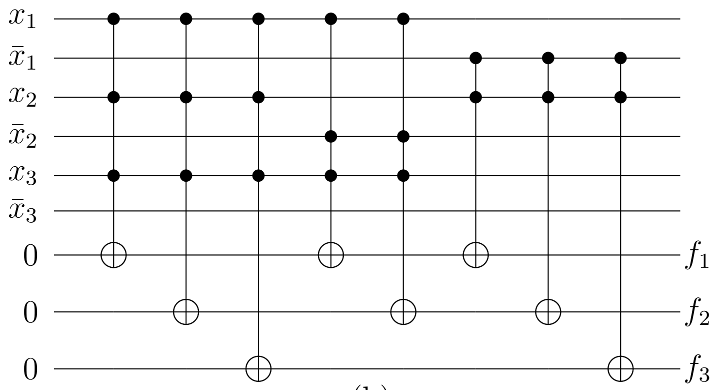
In reversible circuit design ESOP expressions are often written as a cube-list. A cube list is an $n \times m$ matrix with m, the number of product terms in the ESOP expression, and $n = i + j$ where i is the number of input variables and j is the number output variables. Each of the m rows in the matrix are the individual cubes that make up the cube list and represent one of the products from the ESOP expression.

Each cube in the list takes the general form: $x_1 x_2 ... x_i f_1 f_2 ... f_j$, where each of the elements $x_1 .. x_i$ represent an input variables and each element $f_1 .. f_j$ represents an output variable. For each cube in the cube list a 1 is written in cube position x_k , $k \in \{1, 2, ..., i\}$ if the variable x_k is in the ESOP product for that row, a 0 is written if the negation $\bar{x}_k$ is present, and a '$-$' nis written if x_k is not present in that product term for that cube. For each element, f_p, $p \in 1, 2, ...j$, a 1 is written if that output variable contains the product represented by the input portion of the list and a 0 otherwise. See figure *draw figure 1* for an example.

Given an ESOP represented as a cube list in this way *[To add citation: Fazel et al.]* proposed a method of reversible logic synthesis that implements the function as a reversible circuit using only Toffoli gates. In this method an empty cascade with $2i + j$ lines is created. Two input lines are given for every input variable where one line corresponds to the variable x_k, the 2^{nd} line corresponds to its negation, $\bar{x}_k$, and the remaining j lines correspond to the output variables. For every output line f_p, a Toffoli gate is placed with its target on line f_p and a control for that target is placed on the input line x_k if there was a 1 in the cube for the corresponding variable, or on the negation line of x_k if there was a 0 in the cube for x_k, see figure (draw figure 2). This method allows a cube-list to be quickly and efficiently transformed into a reversible cascade and allows for the synthesis of rather large functions.

x_1	x_2	x_3	f_1	f_2	f_3
1	1	1	1	1	1
1	0	1	1	1	0
0	1	$-$	1	1	1

(a)



(b)

Figure 1: A (a) the cube-list and (b) resulting circuit.

5 ESOP Cube-list Ordering Rules

Modifications to the above method with the use of inverters instead of a negation line, careful ordering of cubes to reduce the number of not gates and and sharing cubes between output lines have both reduced the number of lines and the number of gates required to implement these circuits see *[To add citations for these papers]*. In *[to add citation for (nayeem/rice - ordering techniques)]* a number of rules were proposed that manipulate the cube-list representation to create a new cube-list with the same output as the first one, depending on the rule and the state of the cube that the rule is being applied to these rules may increase or decrease the number of cubes in the list.

[To add brief description of ordering rules as these are to be used in our mutation function ...]

References

Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Phys. Rev. A*, 52:3457–3467, Nov 1995. doi: 10.1103/PhysRevA.52.3457. URL http://link.aps.org/doi/10.1103/PhysRevA.52.3457.

David P. DiVincenzo and J. Smolin. Results on two-bit gate design for quantum computers. In *Physics and Computation, 1994. PhysComp '94, Proceedings., Workshop on*, pages 14–23, Nov 1994. doi: 10.1109/PHYCMP.1994.363704.

Martin Lukac, Marek Perkowski, Hilton Goi, Mikhail Pivtoraiko, Chung Hyo Yu, Kyusik Chung, Hyunkoo Jeech, Byung-Guk Kim, and Yong-Duk Kim. Evolutionary approach to quantum and reversible circuits synthesis. *Artif. Intell. Rev.*, 20(3-4):361–417, December 2003. ISSN 0269-2821. doi: 10.1023/B:AIRE.0000006605.86111.79. URL http://dx.doi.org/10.1023/B:AIRE.0000006605.86111.79.

N. M. Nayeem and J. E. Rice. A shared-cube approach to esop-based synthesis of reversible logic. *Facta Universitatis Series: Electronics and Energetics*, 24:385–403, 2011. ISSN 0353-3670.

J.E. Rice and N. M. Nayeem. Ordering techniques for esop-based toffoli cascade generation. In *Communications, Computers and Signal Processing (PacRim), 2011 IEEE Pacific Rim Conference on*, pages 274–279, Aug 2011. doi: 10.1109/PACRIM.2011.6032905.

R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. RevLib: An online resource for reversible functions and reversible circuits. In *Int'l Symp. on Multi-Valued Logic*, pages 220–225, 2008. RevLib is available at http://www.revlib.org.

C.P. Williams. *Quantum Computing and Quantum Communications: First NASA International Conference, QCQC '98, Palm Springs, California, USA, February 17-20, 1998, Selected Papers*. Lecture Notes in Computer Science. Springer, 1999. ISBN 9783540655145. URL http://books.google.ca/books?id=4QuAhlwj2icC.