Hierarchical Reversible Logic Synthesis Using LUTs

Mathias Soeken[1] Martin Roetteler[2] Nathan Wiebe[2] Giovanni De Micheli[1]
[1]Integrated Systems Laboratory, EPFL, Lausanne, Switzerland
[2]Microsoft Research, Redmond, WA, USA

ABSTRACT

Today's rapid advances in the physical implementation of quantum computers demand for scalable synthesis methods in order to map practical logic designs to quantum architectures. We present a synthesis algorithm for quantum computing based on k-LUT networks, which can be derived from Verilog netlists using state-of-the-art and off-the-shelf mapping algorithms. We demonstrate the effectiveness of our method in automatically synthesizing several floating point networks up to double precision. As many quantum algorithms target scientific simulation applications, they can make rich use of floating point arithmetic components. But due to the lack of quantum circuit descriptions for those components, it is not possible to find a realistic cost estimation for the algorithms. Our synthesized benchmarks provide cost estimates that allow quantum algorithm designers to provide the first complete cost estimates for a host of quantum algorithms. This is an essential step towards the goal of understanding which quantum algorithms will be practical in the first generations of quantum computers.

1 INTRODUCTION

Recent progress in fabrication makes the practical application of quantum computers a tangible prospect [11, 19, 26]. However, as quantum computers scale up to tackle problems in computational chemistry, machine learning, and cryptanalysis, design automation will be necessary to fully leverage the power of this emerging computational model.

A major problem facing quantum computing is the inability of existing hand crafted approaches to generate networks for scientific operations that require a reasonable number of quantum bits and gates. As an example, the quantum linear systems algorithm requires as few as 100 (logical) quantum bits to encode a $2^{100} \times 2^{100}$ matrix inversion problem [8, 15]. However, in prior approaches the reciprocal step in the calculation can require in excess of 500 quantum bits which means that arithmetic may dominate the memory requirements (i.e., number of qubits) of that algorithm [32]. Similarly, recent quantum chemistry simulation algorithms can provide improved scaling over the best known methods but at the price of requiring the molecular integrals that define the problem to be computed [4]. While floating point addition was studied before [16, 24], at present networks do not exist for more complex floating point operations such as exponential, reciprocal square root, multiplication, and squaring. Without the ability to automatically generate circuits for these operations it will be a nearly impossible task to estimate the full costs of such algorithms let alone verify that the underlying circuitry is correct.

It has recently been shown [28] that hierarchical reversible logic synthesis methods based on logic network representations are able to synthesize large arithmetic designs. The underlying idea is to map subnetworks into reversible networks. If the subnetworks are small enough, one can use less scalable functional reversible synthesis methods that are based on Boolean satisfiability [14], truth tables [21], or decision diagrams [31]. However, logic networks differ quite significantly from reversible logic networks when considering their structure. This is one of the main disadvantages of currently known hierarchical synthesis methods. As one example, when using reversible circuits in quantum computers, all outputs must either compute a primary input value, a primary output value, or a constant—they cannot expose an intermediate result to an output line, which is referred to as garbage output. State-of-the-art algorithms such as the approach presented in [28] do not explicitly consider techniques to "uncompute values" such that there are no garbage outputs. In order to use the circuit in a quantum computer, one needs to apply a technique called "Bennett trick" [5], which requires to double the number of gates and add one additional circuit line for each primary output.

In this paper we present a hierarchical synthesis approach based on k-feasible Boolean logic networks. These are logic networks in which every gate has at most k inputs. These are often also referred to as k-LUT (lookup table) networks. We show that there is a one-to-one connection between a k-input LUT in a logic network and a reversible single-target gate with k control lines in a reversible network. A single-target gate has a control function and a single target line, that is inverted if and only if the control function evaluates to 1. Each single-target gate can be synthesized into a quantum circuit using techniques such as exclusive-sum-of-product (ESOP) decomposition [13]. As a first step, our synthesis approach can quickly derive a skeleton for the reversible network that is only based on single-target gates. In this skeleton, the number of required additional lines is already final, and also it is guaranteed that it has no garbage outputs. In the second step, each single-target gate is synthesized using a separate algorithm. It is possible to parallelize the second step.

We used our algorithm to find reversible logic networks for several floating point arithmetic networks up to double precision. From these networks we can derive cost estimates for their use in quantum algorithms. This has been a missing information in many proposed algorithms, and arithmetic computation has often not been explicitly taken into account. Our cost estimates show that this is misleading as for some algorithms the arithmetic computation accounts for the dominant cost.

DAC'17, Austin, TX, USA

DOI: 10.1145/3061639.3062261

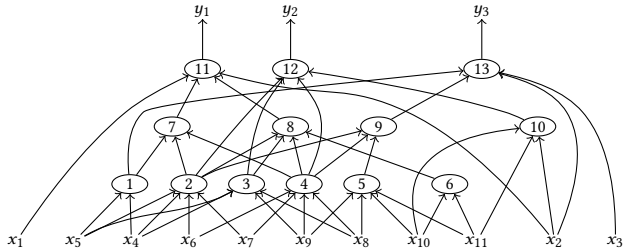

Figure 1: A 4-feasible network with 11 inputs, 3 outputs, and 13 gates

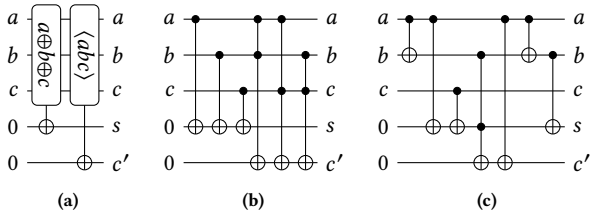

Figure 2: Reversible circuit for a full adder using (a) 2 single-target gates, (b) 3 Toffoli gates and 3 CNOT gates, and (c) 1 Toffoli gate and 6 CNOT gates.

2 PRELIMINARIES

2.1 Some Notation

A digraph $G = (V, A)$ is called *simple*, if $A \subseteq V \times V$, i.e., there can be at most one arc between two vertices for each direction. We refer to $d^-(v) = \#\{w \mid (w, v) \in A\}$ and $d^+(v) = \#\{w \mid (v, w) \in A\}$ as *in-degree* and *out-degree* of v. We use $[n]$ as the short hand for $\{1, \ldots, n\}$.

2.2 Boolean Logic Networks

A Boolean *logic network* is a simple digraph whose vertices are primary inputs, primary outputs, and gates and whose arcs connect gates to inputs, outputs, and other gates. Formally, a Boolean logic network $N = (V, A, F)$ consists of a simple digraph (V, A) and a function mapping F. It has vertices $V = X \cup Y \cup G$ for primary inputs X, primary outputs Y, and gates G. We have $d^-(x) = 0$ for all $x \in X$ and $d^+(y) = 0$ for all $y \in Y$. Arcs $A \subseteq (X \cup G \times G \cup Y)$ connect primary inputs and gates to other gates and primary outputs. Each gate $g \in G$ realizes a Boolean function $F(g) : \mathbb{B}^{d^-(g)} \to \mathbb{B}$. Finally, we call a network *k-feasible* if $d^-(g) \leq k$ for all $g \in G$. Sometimes k-feasible networks are referred to as k-LUT networks (LUT means lookup-table) and LUT mapping (see, e.g., [7, 9, 22, 27]) refers to a family of algorithms that obtain k-feasible networks, e.g., from homogeneous logic representations such as And-inverter graphs (AIGs, [18]) or Majority-inverter graphs (MIGs, [2]).

Example 2.1. Fig. 1 shows a 4-feasible network of the benchmark *cm85a* obtained using ABC [6]. It has 11 inputs, 3 outputs, and 13 gates. The gate functions are not shown but it can easily be checked that each gate has at most 4 inputs.

2.3 Reversible Logic Networks

A reversible logic network realizes a reversible function, which makes it very different from conventional logic networks. Reversible networks are a cascade of reversible gates and the most general gate we consider in this paper is the *single-target gate*. A single-target gate $\mathrm{T}_c(\{x_1, \ldots, x_k\}, x_{k+1})$ has *control lines* $x_1, \ldots, x_k$, a *target line* x_{k+1}, and a *control function* $c : \mathbb{B}^k \to \mathbb{B}$. It realizes the reversible function $f : \mathbb{B}^{k+1} \to \mathbb{B}^{k+1}$ with $f : x_i \mapsto x_i$ for $i \leq k$ and $f : x_{k+1} \mapsto x_{k+1} \oplus c(x_1, \ldots, x_k)$. All reversible functions can be realized by cascades of single-target gates [10]. We use the '$\circ$' operator for concatenation of gates.

Example 2.2. Fig. 2(a) shows a reversible circuit that realizes a full adder using two single-target gates, one for each output. Two additional lines, called *ancilla* and initialized with 0, are added to the network to store the result of the outputs. All inputs are kept as output.

2.4 Mapping to Quantum Circuits

The most commonly used approach to implement quantum circuits is to construct a classical reversible circuit with multiple-controlled Toffoli gates and map these into a sequence of Clifford gates and T gates. A multiple-controlled Toffoli gate is a special single-target gate in which the control function is 1 (tautology) or can be expressed in terms of a single product term. One can always decompose a single-target gate $\mathrm{T}_c(\{x_1, \ldots, x_k\}, x_{k+1})$ into a cascade of Toffoli gates

$$\mathrm{T}_{c_1}(X_1, x_{k+1}) \circ \mathrm{T}_{c_2}(X_2, x_{k+1}) \circ \cdots \circ \mathrm{T}_{c_l}(X_l, x_{k+1}), \qquad (1)$$

where $c = c_1 \oplus c_2 \oplus \cdots \oplus c_l$, each c_i is a product term or 1, and $X_i \subseteq \{x_1, \ldots, x_k\}$ is the support of c_i. This decomposition of c is also referred to as ESOP decomposition. If $c = x_i$, we refer to $\mathrm{T}_c(\{x_i\}, x_{k+1})$ as CNOT gate.

Example 2.3. Fig. 2(b) shows the full adder circuit from the previous example in terms of Toffoli gates. Each single-target gate is expressed in terms of 3 Toffoli gates. Positive and negative control lines of the Toffoli gates are drawn as solid and white dots, respectively. Fig. 2(c) realizes the same output function, albeit with 1 Toffoli gate.

Quantum circuits are described in terms of a small library of gates that interact with one or two qubits. One of the most frequently considered libraries is called the so-called Clifford+T gate library that consists of the reversible CNOT gate, the Hadamard gate, and the T gate. The T-gate is sufficiently expensive in most approaches to fault tolerant quantum computing [3] that it is customary to neglect all other gates when costing a quantum algorithm.

Several works from the literature describe how to map reversible gates into Clifford+T gates (see, e.g., [1, 3, 20]). Note that circuits exist that only require 4 T gates to apply Toffoli up to a phase rotation on the target [17]. While the latter circuits can often be used in place of a standard Toffoli gate, they cannot always be used in this fashion. As such, we focus on the 7 T-gate networks in our synthesis algorithms. Consequently, our costs could be pessimistic by a factor of as much as 7/4. Improvements to the decomposition of multiple-controlled Toffoli gates into Clifford+T circuits have an immediate positive effect on our proposed synthesis method.

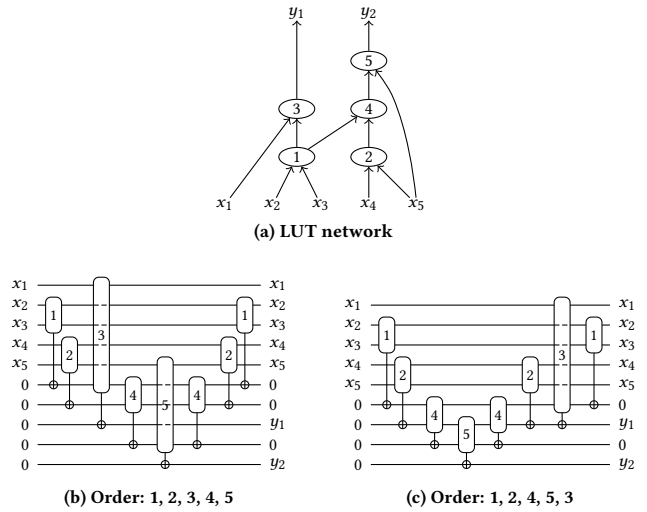

Figure 3: Simple LUT network to illustrate order heuristics (dashed lines in the single-target gates mean that the line is not input to the gate)

3 GENERAL IDEA AND MOTIVATION

This section illustrates the general idea on how to map LUT networks into reversible circuits. For this purpose, take a look at the LUT network in Fig. 3(a). The network consists of 5 inputs $x_1, \ldots, x_5$ and 5 LUTs with names 1 to 5. It has two outputs, y_1 and y_2, which functions are computed by LUT 3 and LUT 5, respectively.

A straightforward way of translating the LUT into a reversible circuit is by using one single-target gate for each LUT in topological order. The target of each single-target gate is a 0-initialized new ancilla line. The reversible circuit in Fig. 3(b) up to the fifth gate results when applying such a procedure. With these five gates, the outputs y_1 and y_2 are realized at line 8 and 10 of the reversible circuit. But after these first five gates, the reversible circuit has garbage outputs on lines 6, 7, and 9 that compute the functions of the inner LUTs of the network. The circuit must be free of garbage outputs in order to be implemented on a quantum computer. This is because the result of the calculation is entangled with the intermediate results and so they cannot be discarded and recycled without damaging the results they are entangled with [25]. We can *uncompute* the intermediate results by re-applying the single-target gates for the LUTs in reverse topological order. This disentangles the qubits, reverting them all to constant 0s. In Fig. 3(b) the last 3 gates uncompute intermediate results at lines 6, 7, and 9. Based on this observation we derive the following lemma.

Lemma 3.1. *When realizing a LUT network with u gates by a reversible circuit that uses single-target gates for each LUT, we need at most u ancilla lines.*

But we can do better. Once we have computed a primary output, we can uncompute LUTs that are not used any longer by other outputs. The uncomputed lines restore a 0 that can be used instead of creating a new ancilla. In the example of Fig. 3, we can first compute output y_2 and then uncompute LUTs 4 and 2, as they are not in the logic cone of output y_1. The freed ancilla can be used for the single-target gate realizing LUT 3. This observation leads to a lemma providing a lower bound.

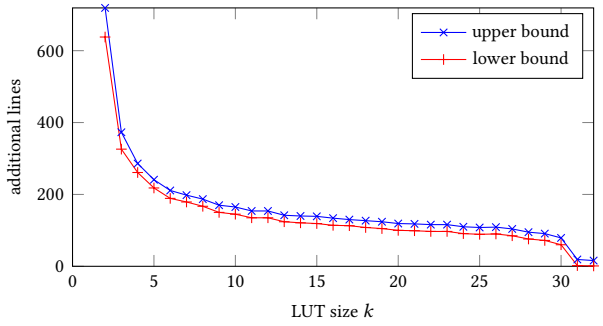

Figure 4: The plot shows the upper and lower bound on the number of additional lines when synthesizing a 16-bit floating point adder from a LUT network with LUT size $k = 2, \ldots, 32$

Lemma 3.2. *Given a LUT network with m outputs, let l be the maximum cone size over all outputs. When realizing the LUT network by a reversible circuit that uses single-target gates for each LUT, we need at least l ancilla lines.*

That is, we start by synthesizing a circuit for the output with the maximum cone. Let's assume that this cone contains l LUTs. They can be synthesized using l single-target gates. From these l gates, $l - 1$ gates can be uncomputed (all except the LUT computing the output), and therefore restores $l - 1$ lines which hold a constant 0 value. We can easily see that the exact number of required lines may be a bit larger, since all output values need to be kept. Further, we may want to make use of logic sharing and use at most two single-target gates for each LUT in the network.

The role of the LUT size. As can be seen from the previous discussion, the number of additional lines roughly corresponds to the number of LUTs. Hence, we are interested in logic synthesis algorithms that minimize the number of LUTs. Several algorithms can be found in the literature [7, 9, 22, 27]. In classical logic synthesis the number of LUT-inputs k needs to be selected according to some target architecture. For example in FPGA mapping, its value is typically 6 or 7. But for our algorithm, we can use k as a parameter that trades off the number of qubits to the number of T gates: If k is small, one needs many LUTs to realize the function, but the small number of inputs also limits the number of control lines in the Toffoli gates obtained from ESOP-based synthesis. And when k is large, one needs fewer LUTs but the resulting Toffoli gates are larger and therefore require more T gates. Further, since for larger k the LUT functions are getting more complex, the runtime potentially increases as ESOP decomposition is becoming more difficult.

To illustrate the influence of the LUT size we performed the following experiment, illustrated in Fig. 4. We applied area-oriented LUT mapping using ABC's [6] command '*if -K k -a*' for $k = 2, \ldots, 32$ to a 16-bit floating point adder. The blue line (×) shows the upper bound according to Lemma 3.1 and the red line (+) shows the lower bound according to Lemma 3.2. First, it can be noted that the bounds are very close to each other. The reason can be that the technology mapping algorithm is efficient in finding shared logic. Second, it

can be seen that for small values of k, up to $k = 6$, the number of additional lines can be reduced significantly. Afterwards, one gains smaller benefit from increasing the cut size. However, for $k = 31$ and $k = 32$, the number of lines can again be significantly reduced. In fact, for $k = 32$, each output can be represented by a single LUT since the adder has 32 inputs.

4 IMPLEMENTATION

The outer structure of the synthesis algorithm is simple. It takes as input a Boolean logic network $N = (V, A, F)$, and outputs a number of lines l of the reversible circuit, and a sequence S of operations. The algorithms manages a map $m : G \to \{0, \ldots, l\}$ that keeps track of which LUT results are computed by which lines. The operations are

- $\texttt{PI}(x, i)$ with an input $x \in X$ and a line $i \in [l]$. This assigns the input x to line i in the circuit.
- $\texttt{PO}(y, i)$ with an output $y \in Y$ and a line $i \in [l]$. This assigns the output y to line i in the circuit.
- $\texttt{COMP}(g, i)$ with a gate $g \in G$ and a line $i \in [l]$. This applies a single-target gate $\mathrm{T}_{F(g)}(\{m(j) \mid j \in \mathrm{fanin}(g)\}, i)$ and sets $m(g) \leftarrow i$.
- $\texttt{UCOMP}(g, i)$ with a gate $g \in G$ and a line $i \in [l]$. This behaves as $\texttt{COMP}(g, i)$ but sets $m(g) \leftarrow 0$.

Example 4.1. The synthesis sequence to produce the circuit in Fig. 3(b) is

$$\texttt{PI}(x_1, 1), \ldots, \texttt{PI}(x_5, 5), \texttt{COMP}(1, 6), \texttt{COMP}(2, 7), \texttt{COMP}(3, 8),$$
$$\texttt{COMP}(4, 9), \texttt{COMP}(5, 10), \texttt{UCOMP}(4, 9), \texttt{UCOMP}(2, 7), \texttt{UCOMP}(1, 6),$$
$$\texttt{PO}(y_1, 8), \texttt{PO}(y_2, 10)$$

Algorithm 1 describes in detail how the synthesis sequence S is obtained. The algorithm keeps track of the current number of lines l, freed lines in a stack C, and a LUT-to-line mapping m (lines 1–3). Also we have a reference counter $r(g)$ for each LUT g that allows us to check when g can be uncomputed. In line 5 and 14, input and output operations are added to S. In between, in lines 6–13, operations for computing and uncomputing gates are determined. Each gate g is visited in topological order. First, the gate g is computed and a 0-initialized line is requested. Either there is one in C or we get a new line by incrementing l. After a COMP operation for g is added to S, we try to uncompute the children recursively by calling uncompute_children. In that function, first the reference counter is decremented for each child g'. If that leads to a reference count of 0, i.e., no other gate needs the computed value of g', we uncompute g' and add the restored line to the stack C. With a given a topological order of LUTs, the time complexity of Algorithm 1 is linear in the number of LUTs.

For the topological order we first compute the cone for each primary output and order them by size in descending order. We perform a topological sort using depth-first search for each cone and do not include duplicates when we visit each cone.

```
   Input  : Logic network N = (V = X ∪ Y ∪ G, A, F)
   Output : Synthesis sequence S, number of lines l
 1 set l ← 1;
 2 initialize empty stack C;
 3 initialize empty map m;
 4 for g ∈ G do set r(g) ← d⁺(g);
 5 for x ∈ X do add PI(x, i) to S, set l ← l + 1;
 6 for g ∈ G in topological order do
 7     set t ← request_constant(C, l);
 8     add COMP(g, t) to S;
 9     set m(g) ← t;
10     if d⁺(g) = 1 and ∃ : y ∈ Y such that (g, y) ∈ A then
11         uncompute_children(g);
12     end
13 end
14 for y ∈ Y do add PO(y, m(g)) to S such that (g, y) ∈ A;
15 return S, l;
16 function request_constant(C, l)
17     if C is not empty then
18         return C.pop();
19     else
20         set l ← l + 1;
21         return l;
22     end
23 function uncompute_children(g, C)
24     for g′ ∈ fanin(g) ∩ G do
25         set r(g′) ← r(g′) − 1;
26         if r(g′) = 0 then
27             add UCOMP(g′, m(g′)) to S;
28             C.push(m(g′));
29             uncompute_children(g′, C);
30         end
31     end
```

Algorithm 1: Obtaining a synthesis sequence

5 EXPERIMENTAL EVALUATION

In the following we refer to our proposed algorithm as *LUT-based Hierarchical Reversible Synthesis* (LHRS). We have implemented the algorithm as command '`lhrs`' on top of the reversible logic synthesis framework RevKit [30].[1] All experiments have been carried out on an Intel Xeon CPU E5-2680 v3 at 2.50 GHz with 64 GB of main memory running Linux 4.4 and gcc 5.4. More details to the benchmarks of the paper and further benchmarks can be found at quantumfpl.stationq.com.

As benchmarks we used Verilog netlists of several arithmetic floating point designs in half (16-bit), single (32-bit), and double (64-bit) precision. For synthesis all Verilog files were translated into AIGs and optimized for size using ABC's '`resyn2`' script. As baseline we compare our results to the state-of-the-art hierarchical reversible logic synthesis algorithm presented in [28], referred to as CBS. CBS partitions an AIG into subnetworks which are then embedded into reversible functions and synthesized using symbolic reversible synthesis algorithms [29]. The size of the subnetworks can be controlled with a threshold parameter t. In our experiments we set t to 10, which results in a similar number of additional lines compared to LHRS with LUT size $k = 6$. It is important to note that CBS does not uncompute results and produces garbage outputs. The reported numbers are based on the circuits with garbage lines, but one can use the "Bennett trick" [5] to uncompute all garbage lines. This trick requires to add one ancilla for each output and double the number of T gates. For CBS we report the number of qubits, an upper bound on the number of T gates according to [20], and the runtime in seconds.

[1] The source code can be found at github.com/msoeken/cirkit

Table 1: Experimental results

Benchmark	CBS [28]			$k = 6$				max k					best k				
	qubits	T gates	runtime	LUTs	qubits	T gates	runtime	k	LUTs	qubits	T gates	runtime	k	LUTs	qubits	T gates	runtime
add-16	287	976840	20.10	211	230	17549	20.53	20	134	156	158389	14.26	14	164	164	111451	14.07
add-32	615	2180502	37.12	480	526	41174	48.69	20	316	368	343849	34.15	14	349	378	132336	32.69
add-64	1323	6099918	128.75	1090	1194	85813	105.93	20	756	867	755391	79.32	7	1090	1117	92709	97.79
cmp-16	71	88847	1.17	33	65	5096	2.37	20	6	38	34879	11.88	15	17	40	75145	0.94
cmp-32	122	101450	1.29	63	126	9256	5.03	20	30	74	991961	146.20	20	30	74	991961	146.20
cmp-64	241	223231	2.85	117	245	17576	8.76	20	63	153	1748784	196.74	8	94	206	33111	6.21
div-16	690	1740176	26.11	275	300	21105	26.55	19	86	112	8378278	18786.70	8	315	258	28285	20.34
div-32	3058	6399903	85.47	1206	1260	66305	107.12	17	851	905	111875120	419368.00	7	1201	1240	69324	113.79
div-64	14481	27514384	350.79	5760	5876	260147	554.79	17	4995	5111	76754630	73753.00	7	5339	5826	267727	534.13
exp-16	1664	3382385	55.02	1356	1371	120113	106.48	16	16	32	1282820	198.17	16	16	32	1282820	198.17
exp-32	5639	10700985	172.48	4605	4636	353926	346.98	18	2907	2938	5796846	11669.20	9	3859	3599	294501	274.80
invsqrt-16	1860	2041683	31.39	1103	899	62887	76.20	16	16	32	214552	9.39	16	16	32	214552	9.39
invsqrt-32	9151	9747362	163.47	4214	4242	190094	338.06	18	3481	3510	90662102	231135.00	6	5387	4242	190094	338.06
invsqrt-64	45815	45581354	799.49	20817	20874	1233350	2025.77	17	19417	19479	196026416	163269.00	7	23442	20646	813404	2140.32
ln-16	1038	5479910	117.24	872	867	90684	68.69	16	16	32	1424454	486.98	16	16	32	1424454	486.98
ln-32	3900	7194487	117.42	3393	3275	263534	250.95	20	873	888	81338102	88226.50	7	3064	3041	263740	232.67
ln-64	26758	36269684	622.92	12298	13149	353642	1188.02	17	11972	11982	14561273	17768.30	17	11972	11982	14561273	17768.30
log2-16	1013	5382593	98.18	953	937	49390	67.07	16	16	32	869010	127.45	16	16	32	869010	127.45
log2-32	4530	7558580	112.36	4001	4008	343082	296.61	18	1124	1127	44250263	154516.00	16	1201	1242	17547121	7794.27
log2-64	12131	21048101	334.23	6318	6414	238135	545.34	20	4842	5749	5985810	3417.62	9	5635	6107	295698	559.45
mult-16	832	2469732	50.01	481	499	47808	41.96	20	249	267	2231718	682.36	15	297	282	944292	79.83
mult-32	2011	7587702	151.58	1499	1536	126158	135.17	20	744	778	6839419	8676.95	19	790	801	6124492	4133.41
mult-64	6860	25813784	496.79	5439	5495	433158	535.25	20	2745	2812	16980088	19791.80	6	5304	5495	433158	535.25
recip-16	1034	3623390	60.20	673	622	58210	60.57	16	16	32	263117	7.27	16	16	32	263117	7.27
recip-32	2866	7646749	125.31	1979	1913	147568	193.97	19	818	840	6071996	6598.64	13	1115	1073	1268452	107.83
recip-64	15919	39149336	641.04	10634	10276	874172	1042.08	20	5892	5391	41490244	87964.90	11	7420	6563	1761316	682.80
sincos-16	435	895815	16.72	190	367	21404	27.83	16	16	33	578680	31.24	16	16	33	578680	31.24
sincos-32	2175	6189192	122.16	1828	1740	132172	140.13	20	698	1228	506492	494.49	9	1454	1483	143937	97.81
square-16	227	569441	9.22	116	113	10219	10.04	16	16	32	196313	6.38	16	16	32	196313	6.38
square-32	1021	2762396	44.96	567	564	38517	49.82	20	178	194	8415178	93561.60	19	179	206	6965063	19863.60
square-64	4674	10821011	154.96	2807	2788	163058	251.47	18	1540	1588	21988382	47948.60	6	2798	2788	163058	251.47
sqrt-16	271	802177	15.59	120	131	11788	9.44	16	16	32	184713	5.61	16	16	32	184713	5.61
sqrt-32	1353	3249386	57.08	704	597	39785	41.89	19	243	274	10024225	15.03	15	552	321	8818533	2707.93
sqrt-64	6793	12560225	194.37	2793	2855	146096	245.43	17	1934	1997	117177325	324895.00	17	1934	1997	117177325	324895.00
sub-16	282	918191	16.72	250	231	17309	20.21	20	146	141	759780	31.56	10	191	185	32947	15.33
sub-32	621	2378121	40.93	562	528	39365	46.07	20	317	337	563680	75.80	15	369	367	191550	32.43
sub-64	1374	6516457	146.30	1172	1191	85225	110.50	20	723	811	917518	92.30	12	837	904	181998	77.02

For LHRS, we used ABC's command '`if -K` k `-a`' to obtain an area-optimized LUT network. Each LUT is decomposed into multiple-controlled Toffoli gates using ESOP-based synthesis on ESOP expressions obtained using ABC's command '`&exorcism`' [23]. We report statistics about the resulting quantum circuits for three different LUT sizes. First, we report $k = 6$, as this is usually the LUT size at which very large LUT count reductions stop (see also Fig. 4). Second, we report a maximum k, called *max k*. We stop the synthesis either if we have successfully found a reversible network based on a network with LUT size $k = 20$ or we hit a timeout limit of 5 days. For large k, generating an initial ESOP cover from a LUT and optimizing the cover using exorcism becomes the bottleneck. Consequently, the runtime typically increases when increasing k. By generating several quantum circuits for different LUT sizes k, we obtain a set of Pareto-optimal solutions. From these one can pick a favorable solution that matches constraints, e.g., imposed by a given architecture or quantum algorithm. To illustrate this, we have plotted the number of qubits and T gates for each k for the 16-bit adder in Fig. 5. As an example for picking a best tradeoff, we chose the largest k before the relative increase in T gates is quintupled. In the example, of the adder, this "sweet spot" is $k = 14$, and we refer to it as *best k*. We list these numbers for all benchmarks in the last five columns in the table.

Note that for all benchmarks with 16 inputs (*exp-16*, *invsqrt-16*, *ln-16*, *log-16*, *recip-16*, *sincos-16*, *square-16*, and *sqrt-16*), the maximum and best k is $k = 16$, because then each output is represented by exactly one LUT and the resulting networks do not need any additional line to store temporary results. All single-target gates have been mapped to Toffoli gates using ESOP decomposition with exorcism [23].

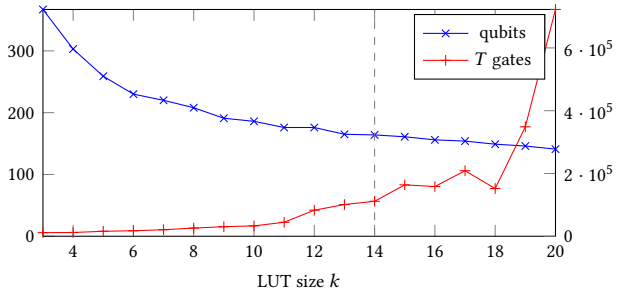

Figure 5: The plot shows all quantum circuit statistics for the benchmark *add-16* for LUT sizes $k = 3, \ldots, 20$. The best LUT size can be found at 14.

It can be seen that LHRS finds quantum circuits with much better qubits/T gates tradeoff. Further, LHRS allows for a better selection of results by using the LUT size as a parameter. One strong advantage is that in LHRS one can quickly obtain a skeleton for the final circuit in terms of single-target gates that already has the final number of qubits. If this number matches the design constraints, one can start the computational more challenging task of finding good quantum circuits for each LUT function. Here, several synthesis passes may be possible in order to optimize the result. Also post-synthesis optimization techniques likely help to signficantly reduce the number of T gates.

6 CONCLUSION

We have provided a new LUT based approach to reversible circuit synthesis that outperforms existing state-of-the-art hierarchical methods such as CBS and unlike such approaches provide networks that are directly applicable to quantum computing. The benchmarks that we provide give what is at present the most complete list of costs for elementary functions for scientific computing. Apart from simply showing improvements, these benchmarks provide cost estimates that allow quantum algorithm designers to provide the first complete cost estimates for a host of quantum algorithms. This is an essential step towards the goal of understanding which quantum algorithms will be practical in the first generations of quantum computers.

While our work provides a meaningful step towards making function synthesis inexpensive for quantum computing, considerable work remains. Two next steps are eminent. First, significant cost improvements can be obtained when using an ESOP decomposition that takes the T gates of the corresponding Toffoli gates into account. Current ESOP decomposition optimizes with respect to the number of product terms, which corresponds to the number of Toffoli gates without considering their different complexities. Second, a LUT mapping algorithm that balances the size of the output cones instead of the overall LUT count can lead to smaller number of qubits.

Acknowledgment. All circuits in this paper were drawn with the qpic tool [12]. This research was supported by H2020-ERC-2014-ADG 669354 CyberCare, the Swiss National Science Foundation (200021-169084 MAJesty), and the ICT COST Action IC1405.

REFERENCES

[1] N. Abdessaied, M. Amy, M. Soeken, and R. Drechsler. Technology mapping of reversible circuits to Clifford+T quantum circuits. In *ISMVL*, pages 150–155, 2016.
[2] L. G. Amarù, P. Gaillardon, and G. De Micheli. Majority-inverter graph: A new paradigm for logic optimization. *IEEE TCAD*, 35(5):806–819, 2016.
[3] M. Amy, D. Maslov, M. Mosca, and M. Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE TCAD*, 32(6):818–830, 2013.
[4] R. Babbush, D. W. Berry, I. D. Kivlichan, A. Y. Wei, P. J. Love, and A. Aspuru-Guzik. Exponentially more precise quantum simulation of fermions in second quantization. *New Journal of Physics*, 18(3):033032, 2016.
[5] C. H. Bennett. Logical reversibility of computation. *IBM Jrnl. of Research and Development*, 17:525–532, 1973.
[6] R. K. Brayton and A. Mishchenko. ABC: an academic industrial-strength verification tool. In *Computer Aided Verification*, pages 24–40, 2010.
[7] D. Chen and J. Cong. DAOmap: a depth-optimal area optimization mapping algorithm for FPGA designs. In *Int Conf on CAD*, pages 752–759, 2004.
[8] B. D. Clader, B. C. Jacobs, and C. R. Sprouse. Preconditioned quantum linear system algorithm. *Physical review letters*, 110(25):250504, 2013.
[9] J. Cong and Y. Ding. FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE TCAD*, 13(1):1–12, 1994.
[10] A. De Vos and Y. Van Rentergem. Young subgroups for reversible computers. *Adv. Math. Comm.*, 2(2):183–200, 2008.
[11] S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536:63–66, 2016.
[12] T. Draper and S. Kutin. $\langle q|pic\rangle$: Creating quantum circuit diagrams in TikZ. *github.com/qpic/qpic*, 2016.
[13] K. Fazel, M. A. Thornton, and J. E. Rize. ESOP-based Toffoli gate cascade generation. In *Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 206–209, 2007.
[14] D. Große, R. Wille, G. W. Dueck, and R. Drechsler. Exact multiple-control Toffoli network synthesis with SAT techniques. *IEEE TCAD*, 28(5):703–715, 2009.
[15] A. W. Harrow, A. Hassidim, and S. Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009.
[16] A. V. Himanshu Thapliyal, Hamid R. Arabnia. Combined integer and floating point multiplication architecture (CIFM) for FPGAs and its reversible logic implementation. *arxiv.org/abs/cs/0610090*.
[17] C. Jones. Low-overhead constructions for the fault-tolerant Toffoli gate. *Physical Review A*, 87(2):022328, 2013.
[18] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai. Robust boolean reasoning for equivalence checking and functional property verification. *IEEE TCAD*, 21(12):1377–1394, 2002.
[19] E. A. Martinez, C. A. Muschik, P. Schindler, D. Nigg, A. Erhard, M. Heyl, P. Hauke, M. Dalmonte, T. Monz, P. Zoller, and R. Blatt. Real-time dynamics of lattice gauge theories with a few-qubit quantum computer. *Nature*, 534:516–519, 2016.
[20] D. Maslov. Advantages of using relative-phase Toffoli gates with an application to multiple control Toffoli optimization. *Phys. Rev. X*, 93:022311, 2016.
[21] D. M. Miller, D. Maslov, and G. W. Dueck. A transformation based algorithm for reversible logic synthesis. In *DAC*, pages 318–323, 2003.
[22] A. Mishchenko, S. Cho, S. Chatterjee, and R. K. Brayton. Combinational and sequential mapping with priority cuts. In *Int Conf on CAD*, pages 354–361, 2007.
[23] A. Mishchenko and M. Perkowski. Fast heuristic minimization of exclusive sum-of-products. In *Int Reed-Muller Workshop*, 2001.
[24] T. D. Nguyen and R. V. Meter. A resource-efficient design for a reversible floating point adder in quantum computing. *JETC*, 11(2):13:1–13:18, 2014.
[25] M. A. Nielsen and I. L. Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.
[26] P. J. J. O'Malley et al. Scalable quantum simulation of molecular energies. *Phys. Rev. X*, 6:031007, 2016.
[27] S. Ray, A. Mishchenko, N. Eén, R. K. Brayton, S. Jang, and C. Chen. Mapping into LUT structures. In *DATE*, pages 1579–1584, 2012.
[28] M. Soeken and A. Chattopadhyay. Unlocking efficiency and scalability of reversible logic synthesis using conventional logic synthesis. In *DAC*, pages 149:1–149:6, 2016.
[29] M. Soeken, G. W. Dueck, and D. M. Miller. A fast symbolic transformation based algorithm for reversible logic synthesis. In *Reversible Computation*, pages 307–321, 2016.
[30] M. Soeken, S. Frehse, R. Wille, and R. Drechsler. RevKit: A toolkit for reversible circuit design. *Multiple-Valued Logic and Soft Computing*, 18(1):55–65, 2012.
[31] M. Soeken, R. Wille, and R. Drechsler. Hierarchical synthesis of reversible circuits using positive and negative Davio decomposition. In *Int Design and Test Workshop*, pages 143–148, 2010.
[32] N. Wiebe and M. Roetteler. Quantum arithmetic and numerical analysis using Repeat-Until-Success circuits. *arXiv:1406.2040*.