Hierarchical Reversible Logic Synthesis Using LUTs

Mathias Soeken\textsuperscript{1}  Martin Roetteler\textsuperscript{2}  Nathan Wiebe\textsuperscript{2}  Giovanni De Micheli\textsuperscript{1}
\textsuperscript{1}Integrated Systems Laboratory, EPFL, Lausanne, Switzerland
\textsuperscript{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 \textit{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 \textit{k}-feasible Boolean logic networks. These are logic networks in which every gate has at most \textit{k} inputs. These are often also referred to as \textit{k}-LUT (lookup table) networks. We show that there is a one-to-one connection between a \textit{k}-input LUT in a logic network and a reversible single-target gate with \textit{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.
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 $F(g) : B^d(g) \rightarrow B$. 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 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 $T_c ((x_1, \ldots, x_k, x_{k+1})$ has control lines $x_1, \ldots, x_k$, a target line $x_{k+1}$, and a control function $c : B^k \rightarrow B$. It realizes the reversible function $f : B^{k+1} \rightarrow B^{k+1}$ 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 $T_c ((x_1, \ldots, x_k))$ into a cascade of Toffoli gates

$$T_{c_1}(x_1, x_{k+1}) \circ T_{c_2}(x_2, x_{k+1}) \circ \cdots \circ T_{c_l}(x_l, x_{k+1}),$$

where $c = c_1 \oplus c_2 \cdots \oplus c_l$, 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 $T_{c_i}(x_{i+1}, x_{k+1})$ 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 $7$-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.
we can uncompute LUTs that are not used any longer by other works into reversible circuits. For this purpose, take a look at the general idea on how to map LUT net-

circuit. But after these first five gates, the reversible circuit has 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_1 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

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, . . . , 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 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-Kk-a’ for k = 2, . . . , 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

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, . . . , 32
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 $m : G \rightarrow \{0, \ldots, l\}$ that keeps track of which LUT results are computed by which lines. The operations are

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

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

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

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.

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]. 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 quantumflx.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 “K 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.
We report statistics about the resulting quantum circuits for three different LUT sizes. First, we report 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 = 6, 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].

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 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 best k.

### Table 1: Experimental results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CBS [28]</th>
<th>k = 6</th>
<th>max k</th>
<th>best k</th>
</tr>
</thead>
<tbody>
<tr>
<td>add-16</td>
<td>287</td>
<td>976840</td>
<td>20.10</td>
<td></td>
</tr>
<tr>
<td>add-32</td>
<td>615</td>
<td>2180502</td>
<td>37.12</td>
<td></td>
</tr>
<tr>
<td>add-64</td>
<td>1322</td>
<td>6099918</td>
<td>128.75</td>
<td></td>
</tr>
<tr>
<td>cmp-16</td>
<td>71</td>
<td>88847</td>
<td>1.17</td>
<td></td>
</tr>
<tr>
<td>cmp-32</td>
<td>122</td>
<td>101450</td>
<td>1.29</td>
<td></td>
</tr>
<tr>
<td>cmp-64</td>
<td>241</td>
<td>223231</td>
<td>2.85</td>
<td></td>
</tr>
<tr>
<td>div-16</td>
<td>690</td>
<td>1704176</td>
<td>26.11</td>
<td></td>
</tr>
<tr>
<td>div-32</td>
<td>3058</td>
<td>6399903</td>
<td>85.47</td>
<td></td>
</tr>
<tr>
<td>div-64</td>
<td>14481</td>
<td>2751414</td>
<td>350.79</td>
<td></td>
</tr>
<tr>
<td>exp-16</td>
<td>1664</td>
<td>3382385</td>
<td>55.02</td>
<td></td>
</tr>
<tr>
<td>exp-32</td>
<td>5639</td>
<td>1070098</td>
<td>172.48</td>
<td></td>
</tr>
<tr>
<td>invsqrt-16</td>
<td>1860</td>
<td>2041683</td>
<td>31.39</td>
<td></td>
</tr>
<tr>
<td>invsqrt-32</td>
<td>9151</td>
<td>9747362</td>
<td>163.47</td>
<td></td>
</tr>
<tr>
<td>invsqrt-64</td>
<td>45815</td>
<td>45581354</td>
<td>799.49</td>
<td></td>
</tr>
<tr>
<td>ln-16</td>
<td>1013</td>
<td>5382593</td>
<td>98.18</td>
<td></td>
</tr>
<tr>
<td>ln-32</td>
<td>3900</td>
<td>7194487</td>
<td>117.42</td>
<td></td>
</tr>
<tr>
<td>ln-64</td>
<td>26758</td>
<td>3626684</td>
<td>622.92</td>
<td></td>
</tr>
<tr>
<td>log2-16</td>
<td>1013</td>
<td>5382593</td>
<td>98.18</td>
<td></td>
</tr>
<tr>
<td>log2-32</td>
<td>4530</td>
<td>7558580</td>
<td>112.36</td>
<td></td>
</tr>
<tr>
<td>log2-64</td>
<td>12131</td>
<td>2104810</td>
<td>334.23</td>
<td></td>
</tr>
<tr>
<td>mult-16</td>
<td>832</td>
<td>2469732</td>
<td>50.01</td>
<td></td>
</tr>
<tr>
<td>mult-32</td>
<td>2011</td>
<td>7587072</td>
<td>151.38</td>
<td></td>
</tr>
<tr>
<td>mult-64</td>
<td>6680</td>
<td>2581378</td>
<td>496.79</td>
<td></td>
</tr>
<tr>
<td>recip-16</td>
<td>1034</td>
<td>3623990</td>
<td>60.20</td>
<td></td>
</tr>
<tr>
<td>recip-32</td>
<td>2866</td>
<td>7646749</td>
<td>125.31</td>
<td></td>
</tr>
<tr>
<td>recip-64</td>
<td>15919</td>
<td>3914936</td>
<td>641.04</td>
<td></td>
</tr>
<tr>
<td>sincos-16</td>
<td>435</td>
<td>895815</td>
<td>16.72</td>
<td></td>
</tr>
<tr>
<td>sincos-32</td>
<td>2175</td>
<td>6189192</td>
<td>122.16</td>
<td></td>
</tr>
<tr>
<td>sincos-64</td>
<td>12741</td>
<td>2104810</td>
<td>334.23</td>
<td></td>
</tr>
<tr>
<td>sqrt-16</td>
<td>271</td>
<td>802177</td>
<td>15.59</td>
<td></td>
</tr>
<tr>
<td>sqrt-32</td>
<td>1535</td>
<td>3293896</td>
<td>75.08</td>
<td></td>
</tr>
<tr>
<td>sqrt-64</td>
<td>6793</td>
<td>12560225</td>
<td>194.37</td>
<td></td>
</tr>
<tr>
<td>sub-16</td>
<td>282</td>
<td>918191</td>
<td>16.72</td>
<td></td>
</tr>
<tr>
<td>sub-32</td>
<td>621</td>
<td>2578121</td>
<td>40.93</td>
<td></td>
</tr>
<tr>
<td>sub-64</td>
<td>1374</td>
<td>6516457</td>
<td>146.30</td>
<td></td>
</tr>
</tbody>
</table>

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 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 table of 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, In-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].
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 significantly 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