Reversible Pebbling Game for Quantum Memory Management

Giulia Meuli* Mathias Soeken* Martin Roetteler† Nikolaj Bjorner† Giovanni De Micheli*
* EPFL, Lausanne, Switzerland †Microsoft, Redmond, WA, USA

Abstract—Quantum memory management is becoming a pressing problem, especially given the recent research effort to develop new and more complex quantum algorithms. The only existing automatic method for quantum states clean-up relies on the availability of many extra resources. In this work, we propose an automatic tool for quantum memory management. We show how this problem exactly matches the reversible pebbling game. Based on that, we develop a SAT-based algorithm that returns a valid clean-up strategy, taking the limitations of the quantum hardware into account. The developed tool empowers the designer with the flexibility required to explore the trade-off between memory resources and number of operations. We present two show-cases to prove the validity of our approach. First, we apply the algorithm to straight-line programs, widely used in cryptographic applications. Second, we perform a comparison with the existing approach, showing an average improvement of 52.77%.

I. INTRODUCTION

The prospective of experimenting with a practical quantum computer is closing up thanks to the recent developments in hardware technology [1], [2]. Driven by the revolutionary potential capabilities of quantum computing, research is extremely active both in academic and in industrial environments. The race is on to develop quantum algorithms capable of proving quantum supremacy, which is the ability to solve problems that cannot be solved classically [3], [4], [5].

A large part of the design of quantum algorithms is still performed manually, despite the emergence of several automatic methods for both synthesis [6], [7] and optimization [8], [9], [10] of quantum circuits. Most manual and automatic approaches for quantum circuit synthesis decompose large functionality into smaller parts in order to deal with complexity. Each part requires some resources in terms of qubits and quantum operations. Most of the parts of a large function are used to compute intermediate values, which are stored on qubits. However, the final circuit must not emit any of those values. Otherwise, the computed results may entangle with intermediate values and compromise the overall quantum algorithm. Since quantum operations are reversible, intermediate results can be “uncomputed” by performing the same operations that computed them, in reverse order. Fig. 1 illustrates a small example. The composition of the two functions \( f \) and \( g \) generates an unknown state that can be uncomputed by performing \( f \) in reverse order.

There are many possible ways to recombine parts of a decomposition, each of which resulting in different accumulated costs for number of qubits and number of quantum operations. The requirement that all intermediate results must be uncomputed makes finding a good strategy particularly difficult in quantum computing. Consequently, effective memory management, which guarantees erasure of intermediate results, is crucial in quantum circuit synthesis.

The problem of finding a strategy to uncompute intermediate states for a fixed number of qubits corresponds to solving the reversible pebbling game. The reversible pebbling game problem has been introduced by Bennett in [11], in the context of exploring space/time trade-off in reversible computation. Input is a Directed Acyclic Graph (DAG), in which each node corresponds to one part of the decomposed computation, and edges define data dependencies. Also, nodes can be pebbled, meaning that the computed value is available on some resource, in our case on a qubit. The game consists of placing pebbles on the graph nodes. Initially no node is pebbled. A pebble can be placed on a node if all its children are pebbled, and the same condition is required to remove a pebble from a node. The game is concluded if all the outputs are pebbled and all the other nodes are unpebbled. Solving the problem returns a valid clean-up strategy. The problem complexity has been studied in [12] where the authors prove that it is PSPACE-complete, as the non-reversible pebbling game. An explicit asymptotic expression for the best time-space product is given in [13], while the asymptotic behavior on trees is studied in [14].

We propose a solution to the reversible pebbling game that casts the problem as a satisfiability problem. We show how the method is capable of exploring the trade-off between space (qubits) and time (operations). In our experimental evaluation, we showcase several examples how our approach can be used to find memory management strategies both for manual and automatic synthesis approaches.

II. PRELIMINARIES

A. Quantum memory management

Our approach abstracts from the actual quantum operations that are being performed, and therefore the interested reader is referred to the literature for a detailed background on quantum computing [15].

The problem of quantum memory management is crucial in quantum circuit design, as all the garbage states need to be carefully cleaned up.

Consider the example of a quantum algorithm that performs the following mapping: \( |x_1\rangle|x_2\rangle|x_3\rangle|x_4\rangle|0\rangle|0\rangle \rightarrow |x_1\rangle|x_2\rangle|x_3\rangle|x_4\rangle|y_1\rangle|y_2\rangle \) where

\[
\begin{align*}
    z_1 &= A(x_2, x_3) \\
    z_2 &= C(z_1, x_3) \\
    z_3 &= B(x_3, x_4) \\
    z_4 &= D(z_3, x_3) \\
    y_1 &= E(z_2, z_4) \\
    y_2 &= F(x_1, z_1)
\end{align*}
\]

with \( A, B, C, D, E, F \) being some generic 2-input Boolean operations and \( z_1, z_2, z_3, z_4 \) the intermediate results. Such computation corresponds to the DAG in Fig. 2. In order to build the quantum circuit to perform our computation we exploit the
A simple solution is the one proposed in Fig. 3(a), which is referred to as the Bennett [11] strategy. It consists of computing all the operations in a bottom-up order, and then uncomputing the intermediate results in a top-down fashion, so that all the nodes have their inputs available. This strategy always leads to the minimum number of gates, and to the maximum number of qubits. The order in which the DAG is converted into a reversible circuit can have a significant effect on how the memory is managed. In the example strategy in Fig. 3(b) it is shown how, by changing the order of the operations, it is possible to save one qubit, without increasing the duration of the computation. Finally, by allowing an increase in the number of gates, we can further reduce the number of ancillae to 4. In this case some functions are computed several times, see Fig. 3(c). In Fig. 3 ancillae are colored red during the time they are storing an intermediate result. The first two strategies store values for a long time during which they are not needed, whereas the last strategy makes a good usage of fewer memory. We cannot state that the last method performs better than the first one, as that would depend from the actual hardware constraints. What we achieve in this work is to empower the designer with the ability to choose whether exchange memory for time and vice versa.

B. Reversible pebbling game

The problem of finding the best uncomputing strategy is equivalent to the reversible pebbling game problem. In the remainder we use independently pebbling and uncomputing strategy.

Definition 2 (Reversible pebbling configuration): A reversible pebbling configuration of a DAG $G = (V, E)$ is the set $P \subseteq V$ of all the pebbled vertices.

Definition 3 (Reversible pebbling strategy): A reversible pebbling strategy of a DAG $G$ is a sequence of reversible pebbling configurations $P = (P_1, \ldots, P_m)$ such that $P_1 = \emptyset$ and $P_m = O$, where $O$ is the set of all sinks of $G$. For each $1 \leq i \leq m$, we have $P_i = P_{i-1} \cup \{v\}$ or $P_i = P_{i-1}/\{\}$ and $P_i \neq P_{i-1}$. All in-neighbors of $v$ are in $P_{i-1}$.

III. SAT-ENCODING

In this work we aim at finding a good pebbling strategy while constraining the maximum number of pebbles per step. Thus, as we want an efficiently pebbling and uncomputing strategy. We impose satisfiability constraints.

Problem 1: Given a DAG and a number of pebbles, find a valid pebbling strategy using the minimum number of steps. As we use a SAT solver [16] to extract our solution, we have to decompose this problem into many SAT problems.

Problem 2: Given a DAG and $K$ pebbles, does a valid pebbling strategy with $K$ steps exist?

The solver can either find a solution and return a pebbling strategy, or state that no solution exists. In this case we increase the number of steps to $K + 1$ until a satisfying solution is found.

Following the definition of a reversible pebbling game given in Section II-B, we first declare our set of variables, and then impose satisfiability constraints.

Variables: The input DAG $G = (V, E)$ has some nodes which compute an output value and we refer to them as a set $O \subseteq V$. Note that the primary inputs are not nodes in the DAG. We also define $C(v) = \{w \mid w \rightarrow v\}$ as all children of a node $v$. Problem 2 is encoded in terms of the pebble state variables $p_{v,i}$. For $v \in V$ and $0 \leq i \leq K$, those are Boolean variables that evaluate to true if the node $v$ is pebbled at time $i$.
with \( K \) steps describing the transition from one configuration to the other.

**Clauses:** The following set of clauses describes the reversible pebbling problem.

**Initial and final clauses** At time 0 all the nodes are unpebbled and at time \( K \) all the outputs need to be pebbled and all the intermediate results unpebbled,

\[
\bigwedge_{v \in V} \bar{p}_{v,0} \land \bigwedge_{v \in O} \bigwedge_{w \in \overline{O}} \bigwedge_{v \in \overline{O}} \bar{p}_{v,K} \land \bigwedge_{v \in \overline{O}} \bar{p}_{v,K}
\]

**Move clauses** If a node is pebbled or unpebbled at time \( i + 1 \), then all its children are pebbled at time \( i \) and time \( i + 1 \):

\[
\bigwedge_{v \in V} \bigwedge_{(v, w) \in E} \left( (p_{v, i} \oplus p_{v, i+1}) \rightarrow (p_{w, i} \land p_{w, i+1}) \right)
\]

**Cardinality clauses** At each step, at most \( P \) pebbles are used:

\[
\bigwedge_{i=0}^{K} \left( \sum_{v \in V} p_{v, i} \leq P \right)
\]

IV. APPLICATIONS AND RESULTS

In this section we illustrate the validity of our proposed approach by show-casing several examples in which large computations are expressed in terms of a sequence of smaller ones. In order to optimally exploit qubit resources, a high-quality quantum memory management is required that can be addressed using our SAT-based reversible pebble game solver. Our algorithm uses at its core the open source SAT solver Z3 [16].

**Straight-line programs:** We apply our method to the synthesis of straight-line programs used in cryptographic applications. Those programs are combinations of modular arithmetic operations as addition, subtraction, multiplication, and squaring. We assume that for each operation a quantum implementation exists, and will have a given cost in terms of quantum gates and ancillae. We can estimate the cost of an algorithm implementation in terms of number of different operations, according to the resources available. We choose a straight-line program that implements the addition between two points of an Edwards curve in projective coordinates from [17].

We pebble the resulting DAG using different number of pebbles. Fig. 4 visualizes the pebbling strategies obtained with 24, 20, 16, 12, and 10 pebbles. In each case, we obtain a different number of operations, as reported in Fig. 4. For example the first implementation performs a total of 74 operations: 28 additions, 20 subtractions, 15 squaring and 11 multiplication. We can see how the tool manages to fit the desired computation into limited number of qubits, by increasing the number of required steps. As a consequence, the last implementation has an higher cost in terms of operations: 110 in total. The overall cost of the algorithm on different hardware can be evaluated having some estimates of the real cost of each operation. On the top of each grid, we show the dynamic change in the memory employed during the computation. A flat dynamic suggests that a constant number of qubits is used through the whole computation. A solution with a lower peak requires less qubits.

**Comparison with Bennett strategy.** The second show-case wants to test the program on the mapping of a design with limited number of qubits. We consider an operator called \( H \) (different from the Hadamard gate) that is used internally to the algorithm that computes the doubling of two points.

### Table I. Comparison

<table>
<thead>
<tr>
<th></th>
<th>pi</th>
<th>po</th>
<th>nodes</th>
<th>Bennett</th>
<th>Pebbling strategy</th>
<th>P</th>
<th>K</th>
<th>runtime</th>
<th>%P × K</th>
</tr>
</thead>
<tbody>
<tr>
<td>b2_m3</td>
<td>8</td>
<td>8</td>
<td>76</td>
<td>124</td>
<td>30</td>
<td>186</td>
<td>0.17</td>
<td>54.55</td>
<td>1.5</td>
</tr>
<tr>
<td>b3_m4</td>
<td>12</td>
<td>12</td>
<td>59</td>
<td>47</td>
<td>82</td>
<td>20</td>
<td>117</td>
<td>37.45</td>
<td>1.43</td>
</tr>
<tr>
<td>b4_m5</td>
<td>16</td>
<td>16</td>
<td>203</td>
<td>187</td>
<td>358</td>
<td>83</td>
<td>778</td>
<td>55.75</td>
<td>2.17</td>
</tr>
<tr>
<td>b5_m7</td>
<td>20</td>
<td>20</td>
<td>256</td>
<td>236</td>
<td>452</td>
<td>106</td>
<td>888</td>
<td>31.15</td>
<td>1.96</td>
</tr>
<tr>
<td>b6_m7</td>
<td>24</td>
<td>24</td>
<td>310</td>
<td>286</td>
<td>548</td>
<td>130</td>
<td>1132</td>
<td>35.72</td>
<td>2.05</td>
</tr>
<tr>
<td>b8_m7</td>
<td>32</td>
<td>32</td>
<td>422</td>
<td>390</td>
<td>748</td>
<td>187</td>
<td>1884</td>
<td>11.59</td>
<td>2.05</td>
</tr>
<tr>
<td>b10_m7</td>
<td>40</td>
<td>40</td>
<td>535</td>
<td>495</td>
<td>950</td>
<td>254</td>
<td>2938</td>
<td>28.66</td>
<td>4.07</td>
</tr>
<tr>
<td>b12_m7</td>
<td>48</td>
<td>48</td>
<td>646</td>
<td>598</td>
<td>1114</td>
<td>331</td>
<td>4228</td>
<td>56.33</td>
<td>4.65</td>
</tr>
<tr>
<td>c17</td>
<td>5</td>
<td>2</td>
<td>12</td>
<td>7</td>
<td>12</td>
<td>4</td>
<td>12</td>
<td>0.01</td>
<td>42.86</td>
</tr>
<tr>
<td>c432</td>
<td>36</td>
<td>7</td>
<td>208</td>
<td>172</td>
<td>337</td>
<td>60</td>
<td>685</td>
<td>23.70</td>
<td>65.12</td>
</tr>
<tr>
<td>c499</td>
<td>41</td>
<td>32</td>
<td>237</td>
<td>178</td>
<td>324</td>
<td>77</td>
<td>610</td>
<td>60.08</td>
<td>56.74</td>
</tr>
<tr>
<td>c880</td>
<td>60</td>
<td>26</td>
<td>334</td>
<td>274</td>
<td>522</td>
<td>82</td>
<td>1280</td>
<td>43.52</td>
<td>70.07</td>
</tr>
<tr>
<td>c1355</td>
<td>41</td>
<td>32</td>
<td>219</td>
<td>178</td>
<td>324</td>
<td>77</td>
<td>594</td>
<td>2.63</td>
<td>56.74</td>
</tr>
<tr>
<td>c1908</td>
<td>33</td>
<td>25</td>
<td>220</td>
<td>187</td>
<td>349</td>
<td>70</td>
<td>875</td>
<td>57.97</td>
<td>62.57</td>
</tr>
<tr>
<td>c2670</td>
<td>157</td>
<td>63</td>
<td>554</td>
<td>397</td>
<td>731</td>
<td>160</td>
<td>1948</td>
<td>47.94</td>
<td>59.7</td>
</tr>
<tr>
<td>c3540</td>
<td>50</td>
<td>22</td>
<td>856</td>
<td>806</td>
<td>1590</td>
<td>416</td>
<td>5434</td>
<td>111.20</td>
<td>48.39</td>
</tr>
<tr>
<td>c3515</td>
<td>178</td>
<td>123</td>
<td>1257</td>
<td>1079</td>
<td>2035</td>
<td>498</td>
<td>7635</td>
<td>118.38</td>
<td>53.85</td>
</tr>
<tr>
<td>c6288</td>
<td>32</td>
<td>32</td>
<td>1011</td>
<td>979</td>
<td>1926</td>
<td>640</td>
<td>10232</td>
<td>101.31</td>
<td>34.63</td>
</tr>
<tr>
<td>c7552</td>
<td>207</td>
<td>108</td>
<td>1151</td>
<td>944</td>
<td>1780</td>
<td>540</td>
<td>7757</td>
<td>124.11</td>
<td>42.38</td>
</tr>
</tbody>
</table>

Average percentage reduction of pebbles = 52.77%
Average multiplicative factor for the number of steps = 2.68

referred before [17]. This operator is a composition of modular additions (+) and modular subtractions (−); it has \( a, b, c, d \) as inputs and four outputs \( x, y, z, t \), where:

\[
t_1 = a + b \quad t_2 = c + d \quad t_3 = a - b \quad t_4 = c - d
\]

\[
x = t_1 + t_2 \quad y = t_1 - t_2 \quad z = t_3 + t_4 \quad t = t_3 - t_4
\]

Experiments reported in Table IV show a comparison with the Bennett pebbling method. The different designs correspond to the \( H \) operator with different bitwidths and modulus. We also report our results for the well known ISCAS benchmark. The graph representation for the function has been extracted from an XOR-majority graph using the open source tool mockturtle [18]. A method to use XOR-majority graphs into quantum circuits using a naive quantum memory management strategy was presented in [19]. The number of pebbles corresponds to the minimum one for which the solver could find a solution within 2 minutes. Even with this restricted timeout the algorithm finds a solution for a significantly reduced number of pebbles. The average percentage reduction is 52.77%. As the pebbles are reduced, the number of steps increase with respect to the Bennett method. In average the number of steps for the constrained design is 2.68× the one of the naive strategy. With the increase of the size of the DAG, we see a fewer pebble reduction. The reason is in the timeout chosen, as the solver takes more time for large designs; the number of variables of the SAT problem is proportional to \( n^2 \) where \( n \) is the number of nodes of the DAG. Also increasing the number of nodes, more steps seems to be required. This is also dependent on the timeout. In fact the algorithm is capable of finding many solutions with different number of pebbles but same number of steps. Nevertheless more constrained solutions require more time to be resolved.

V. CONCLUSION

We developed a SAT-based algorithm for quantum memory management. We show how the clean-up problem corresponds to the reversible pebbling game problem. Consequently, our algorithm solves instances of the reversible pebbling game to explore the trade-off between memory and number of operations. Finding an efficient pebbling strategy is crucial in quantum algorithm development, where often small manually optimized circuits are cascaded together. Our tool can enable computations in a constrained system, when this would not
be possible using the strategies in the literature. We have shown two different show-cases to demonstrate the efficiency of our method. In general, it can be used in cryptographic applications to synthesize straight-line programs, but also in any hierarchical synthesis automatic method. It can be used to estimate the cost of performing an algorithm on a given hardware in terms of number of operations. Our experiments show that we are capable of finding solutions with an average reduction in number of ancillae required of 52.77% with a timeout of 2 minutes. Finally, the tool could be used by a designer to map a required computation into the available hardware.

Acknowledgments: This research was supported by H2020-ERC-2014-ADG 669354 CyberCare and the Swiss National Science Foundation (200021-169084 MAJesty and and 200021-146600).

REFERENCES