Efficient Construction of Functional Representations for Quantum Algorithms

Lukas Burgholzer[1], Rudy Raymond[2], Indranil Sengupta[3], and Robert Wille[1,4]

[1] Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
[2] IBM Quantum, IBM Research - Tokyo, Japan
[3] Indian Institute of Technology Kharagpur, India
[4] Software Competence Center Hagenberg GmbH (SCCH), 4232 Hagenberg, Austria
{lukas.burgholzer,robert.wille}@jku.at rudyhar@jp.ibm.com isg@iitkgp.ac.in
https://iic.jku.at/eda/research/quantum

Abstract. Due to the significant progress made in the implementation of quantum hardware, efficient methods and tools to design corresponding algorithms become increasingly important. Many of these tools rely on functional representations of certain building blocks or even entire quantum algorithms which, however, inherently exhibit an exponential complexity. Although several alternative representations have been proposed to cope with this complexity, the *construction* of those representations remains a bottleneck. In this work, we propose solutions for *efficiently constructing* representations of quantum functionality based on the idea of conducting as many operations as possible on as small as possible intermediate representations—using Decision Diagrams as a representative functional description. Experimental evaluations show that applying these solutions allows to construct the desired representations several factors faster than with state-of-the-art methods. Moreover, if repeating structures (which frequently occur in quantum algorithms) are explicitly exploited, exponential improvements are possible—allowing to construct the functionality of certain algorithms within seconds, whereas the state of the art fails to construct it in an entire day.

1 Introduction

Quantum computing promises to outperform classical computers in certain applications. While the theoretical background was already developed in the previous century, it is today that actual physical devices are evolving to a point where first experiments are performed that are suggested not to be easy on a classical computer. However, having hardware without efficient tools to design corresponding algorithms on it certainly presents an unsatisfactory situation. Accordingly, researchers and engineers started to develop methods and tools for important tasks such as synthesis/compilation [1]–[5], (classical) simulation [6]–[8], and verification [9]–[12]—leading to elaborate design flows and tool chains as realized, e.g., by IBM's Qiskit [13], Google's Cirq [14], and Microsoft's QDK [15].

These tools and the corresponding design tasks, however, frequently rely on representations of certain building blocks' functionality or even the functionality of an entire quantum algorithm. This poses a severe challenge since quantum functionality is most generally described by matrices of exponential dimension with respect to the size of the quantum system, i.e., $2^n \times 2^n$ for a system consisting of n qubits (the quantum analogue to bits). To date, industrial tool chains like IBM's Qiskit hardly offer efficient and scaleable solutions for constructing and representing quantum functionality (as witnessed by the evaluations later in Section 5).

Fortunately, different approaches have been proposed that try to deal with this complexity, e.g., based on arrays [16]–[19], tensor networks [20]–[23], and Decision Diagrams [24]–[26]. Although we may be able to represent (i.e., store) the overall functionality of certain building blocks or an entire quantum algorithm using these techniques, we may not be able to construct this representation in feasible time—which constitutes a severe bottleneck for many applications in the domain of quantum computing. This is caused by the fact that, even though individual quantum operations typically emit a sparse, tensor product structure, their composition requires subsequent *matrix-matrix* multiplications—leading to a potential decrease in sparsity and/or exploitable structure. Hence, many computations on potentially large intermediate representations have to be conducted in order to construct the overall functional representation.

In this paper, we propose two solutions to overcome this bottleneck—using Decision Diagrams (DDs) as a representative functional description. First, a general solution is presented which can be applied to arbitrary functionality and is based on the idea to conduct as many operations as possible on as small as possible intermediate representations. Besides that, another solution is proposed which explicitly exploits the fact that many quantum algorithms contain repeating structures (e.g., Grover iterations, random walks, etc.). In both cases, the complexity of constructing quantum functionality representations is substantially reduced—in case of the second solution even an exponential improvement is achieved.

Experimental evaluations eventually confirm the resulting benefits. They show that the proposed solutions allow to construct the desired representations several factors faster than with the current state of the art. If additionally repeating structures are exploited, representations for quantum algorithms and building blocks can be constructed in a matter of seconds which, using the current state of the art, could not be constructed in an entire day. The resulting implementation is available as open source at `https://github.com/iic-jku/qfr`.

The rest of this paper is structured as follows: Section 2 reviews the necessary basics on quantum computing and introduces the Quantum Fourier Transform, which will be used as a running example in this paper. In Section 3, we show the importance of the considered problem and review the state of the art—illustrating the current bottleneck. Then, Section 4 introduces and describes the proposed solution which, afterwards, is evaluated in Section 5. Finally, Section 6 concludes the paper.

2 Background

In this section, we briefly review the key concepts of quantum computing as well as a typical building block for quantum algorithms which will serve as an example over the course of this paper. While the respective reviews are kept brief, we refer the interested reader to [27] for a more thorough treatment on quantum computing.

In classical computing, *bits* are used as the smallest computation unit—attaining values from the discrete set $\mathbb{B} = \{0, 1\}$. In the field of quantum computing, these discrete values, denoted $|0\rangle$ and $|1\rangle$ using Dirac notation, are chosen as basis elements spanning a two-dimensional complex Hilbert space $\mathbb{H}$. Consequently, the state $|q\rangle$ of a *qubit* (the quantum analogue to the bit) is described by an element of this space, i.e., by a *superposition* of the basis states $|0\rangle$ and $|1\rangle$. More specifically, $|q\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$ with $\alpha_i \in \mathbb{C}$ such that $|\alpha|^2 = |\alpha_0|^2 + |\alpha_1|^2 = 1$.

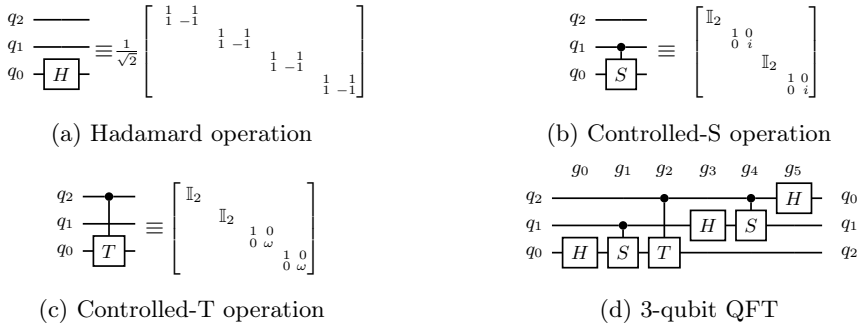
A *quantum system* then consists of n qubits $q_0, \ldots, q_{n-1}$ described by the 2^n-dimensional Hilbert space $\mathbb{H} \otimes \cdots \otimes \mathbb{H}$. The state $|q\rangle_n$ of such a system is again described by amplitudes $\alpha_i \in \mathbb{C}$, where $|q\rangle_n = \sum_{i \in \{0,1\}^n} \alpha_i |i\rangle$ with $|\alpha|^2 = 1$. However, the amplitudes α_i of a quantum system are not directly observable. Instead, performing a *measurement* probabilistically collapses the qubits' state to one of the basis states $|i\rangle$ (each with probability $|\alpha_i|^2$).

The state of a quantum system is manipulated through unitary linear transformations $U\colon\ \mathbb{H} \otimes \cdots \otimes \mathbb{H} \to \mathbb{H} \otimes \cdots \otimes \mathbb{H}$, which are predominantly described by their unitary $2^n \times 2^n$ matrix representations[5] in the computational basis $\{|0\rangle, \ldots, |2^n - 1\rangle\}$. Usually, these *quantum operations* act only on a small subset of a system's qubits. Hence, their matrix representations have a sparse, tensor product structure, where the tensor product of smaller "operation matrices" with identity matrices is formed.

Example 1. Consider a quantum system consisting of $n = 3$ qubits. Then, Fig. 1a, Fig. 1b, and Fig. 1c show a few common quantum operations using their $2^3 \times 2^3$ sparse matrix representations—namely the Hadamard operation as well as the controlled-phase operations S and T, where $\omega = \exp(\frac{2\pi i}{8}) = \sqrt{i}$.

A *quantum algorithm* is described as a sequence of quantum operations applied to a quantum system, i.e., $G = g_0, \ldots, g_{m-1}$ denotes a quantum algorithm consisting of m operations where each g_i is described by a unitary matrix U_i. Since the composition of unitary transformations is again unitary, the functionality of a quantum algorithm may be interpreted as one unitary transformation itself. Consequently, the functionality is described by a unitary matrix U which arises from the *matrix-matrix* multiplication of the individual operation matrices U_i, i.e., $U = U_{m-1} \cdots U_0$. Quantum algorithms are usually visualized as *quantum circuit diagrams* where wires indicate the individual qubits and operations (also called *gates*) are placed as boxes on these lines with corresponding identifiers. Time is assumed to progress from left to right.

[5] A complex-valued matrix U is unitary if $U^\dagger U = U U^\dagger = \mathbb{I}$, where $U^\dagger$ denotes the conjugate transpose of U and $\mathbb{I}$ the identity matrix.



(a) Hadamard operation

(b) Controlled-S operation

(c) Controlled-T operation

(d) 3-qubit QFT

Fig. 1: Common quantum operations and the QFT in a 3-qubit system

In the following, quantum algorithms and quantum circuit diagrams will be illustrated by means of the *Quantum Fourier Transform* (QFT) [27]—a well-known building block in many important quantum algorithms. Its most prominent use probably is for period finding in Shor's algorithm for integer factorization [28] and many other group-theoretic problems (see Chapter 5 of [27]), at which exponential speed-ups over the best classical methods are demonstrated. QFT is also used in quantum approximate counting [29], which provides proven polynomial speed-ups over the best classical methods, i.e., Monte-Carlo-type estimators [30]. Such quantum Monte-Carlo algorithms are now popular candidates to achieve quantum advantage with near-term quantum devices [31].

Example 2. Consider a 3-qubit system as already discussed in Example 1. Then, Fig. 1d shows the quantum circuit for the 3-qubit Quantum Fourier Transform consisting of $m = 6$ gates in total. This circuit will be used as a running example for the further discussions throughout this work.

3 Representations for Quantum Algorithms

Working in the domain of quantum computing requires representations of certain building blocks or even entire quantum algorithms. This is evident, e.g., for typical tasks such as:

- *Synthesis/Compilation* [1]–[5], where an entire quantum algorithm is realized in terms of elementary quantum operations supported by the addressed quantum architecture. Without a proper representation of the algorithm's functionality, no synthesis/compilation approach can work.
- *(Classical) Simulation* [6]–[8], where a given quantum algorithm is "tested" on a classical machine prior to actual execution on a quantum computer. While this can be done using consecutive matrix-vector multiplication on the elementary gates, approaches based on *emulation* [32], [33], which utilize functional representations of entire building blocks, have been shown to be much more efficient—provided the emulated functionality can be constructed efficiently.

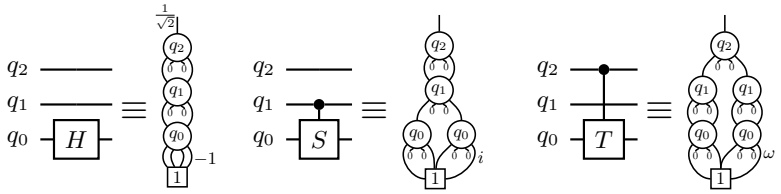


Fig. 2: Decision Diagrams for operations shown in Fig. 1a—1c

– *Verification* [9]–[12], where, e.g., for two quantum circuits G and G' it should be checked whether they realize the same function—also referred to as *equivalence checking*. This obviously requires the construction of a functional representation for both functionalities in order to compare them.

In all these cases, having a representation of the considered functionality is essential. The first challenge resulting from that is that quantum functionality in general is described in terms of matrices with exponential size, i.e., for a functionality over n qubits, a matrix U of size $2^n \times 2^n$ results. In previous work, researchers already started to address this challenge, which led to different approaches exploiting certain structural elements of the considered functionality in order to reduce the exponential space complexity of its representation:

– *Array-based approaches* (such as proposed in [16]–[19]) heavily rely on the sparsity of the involved matrices and try to distribute the workload over several cores of supercomputers, which can often be done efficiently since the *matrix-multiplication* itself is inherently parallelizable.
– *Tensor Networks* (such as proposed in [20]–[23]) capitalize on the tensor product structure inherent to quantum operations—allowing to decompose the whole matrix into many smaller parts. Their performance typically scales with the degree of entanglement of the considered functionality.
– *Decision Diagrams* (DDs, such as proposed in [24]–[26]) recursively split the considered functionality into equally sized sub-matrices until only complex numbers remain. By identifying redundancies in these sub-parts and extracting common factors, equal sub-functionality can be shared—frequently leading to a compact representation in terms of directed acyclic graphs with edge-weights.

In the following, we will illustrate those endeavours using Decision Diagrams as a representative. However, the observations and findings discussed in this work apply to the other representations as well.

Example 3. Consider again the quantum operations shown in Fig. 1. Their functionalities can be represented efficiently in terms of Decision Diagrams as shown in Fig. 2. As can be seen, they allow for a rather compact representation (3–5 nodes vs. 8–16 non-zero matrix entries).

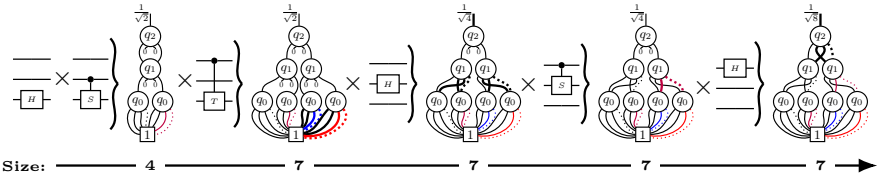


Fig. 3: State-of-the-art DD composition sequence for 3-qubit QFT (see Fig. 1d)

Unfortunately, *constructing* those representations for certain building blocks or even entire quantum algorithms can often not be conducted efficiently—even if it is conceptionally simple[6]. In fact, as reviewed in Section 2, the functionality of a quantum algorithm (given by a quantum circuit $G = g_0, \ldots, g_{m-1}$) is described by the matrix $U = U_{m-1} \cdots U_0$, with U_i being the matrix corresponding to gate g_i (for $0 \leq i < m$). Hence, since the individual matrices U_i can usually be represented rather efficiently with either of the approaches reviewed above (arrays, tensor networks, DDs), simply conducting multiplications on those representations should allow for an efficient construction of the entire functional representation. But the more quantum operations are multiplied together, the more complex representations result—reducing the sparsity, increasing the degree of entanglement, or eliminating existing redundancies—and, hence, significantly slowing down the construction. Thus, while the multiplication operation itself is realized rather efficiently in general (utilizing, e.g., specialized techniques for sparse chain multiplication), the bottleneck arises from the consequences of consecutive multiplication.

Example 4. Consider again the circuit for the 3-qubit QFT from Fig. 1d. Constructing its functionality requires multiplying the individual representations of all 6 gates. The multiplication of two Decision Diagrams (representing matrices U and V) is recursively broken down into sub-expressions according to

$$\begin{bmatrix} U_{00} & U_{01} \\ U_{10} & U_{11} \end{bmatrix} \cdot \begin{bmatrix} V_{00} & V_{01} \\ V_{10} & V_{11} \end{bmatrix} = \begin{bmatrix} (U_{00}V_{00} + U_{01}V_{10}) & (U_{00}V_{01} + U_{01}V_{11}) \\ (U_{10}V_{00} + U_{11}V_{10}) & (U_{10}V_{01} + U_{11}V_{11}) \end{bmatrix},$$

until only operations on complex numbers remain. That results in a complexity which scales with the product of the number of nodes in the Decision Diagrams to be multiplied. Carrying out all multiplications results in the evolution of representations as shown in Fig. 3[7]. While, as already shown by means of Fig. 2, the functionality of single operations can be represented compactly, the multiplication needed to construct the overall functionality quickly increases the complexity. In fact, after two multiplications, further computations have to be conducted on a representation as large as the final result.

[6] The authors want to point out that this construction task is conceptionally different from and should not be confused with the classical simulation of quantum circuits which aims to calculate the resulting state vector for one particular input and not the complete functionality.

[7] Different edge weights are indicated by dotted (≡ negative) and/or colored (≡ 1, i, ω and ω^3) lines. This suffices to illustrate the evolution of the Decision Diagrams' size, i.e., their node count.

Evaluations on larger examples than the one above confirm that, in many cases, we may be able to represent (i.e., store) the overall functionality of certain building blocks or an entire quantum algorithm (and use it for tasks such as synthesis/compilation, simulation, or verification), but we may not be able to construct this representation in feasible time. This constitutes a severe bottleneck for many applications in the domain of quantum computing.

4 Proposed Approaches

In order to overcome the bottleneck discussed in the previous section, we propose to approach the construction of functional representations for building blocks or entire quantum algorithms with different strategies. We distinguish thereby two use cases: First, a general construction scheme is presented which can be applied for arbitrary functionality. Afterwards, we present a second scheme which is dedicated to repeating structures as they frequently occur in quantum algorithms (e.g., by means of Grover iterations or quantum walks). The resulting schemes allow to speed up the construction of the desired functional representation considerably and even manage to complete the construction where existing methods time out.

4.1 General Scheme

The observations from Section 3 show that the bottleneck emerges as a result of a large number of matrix-matrix multiplications on rather large representations. Hence, in order to avoid this, we propose to conduct as many of those multiplications on as small as possible representations, e.g., on the original gate representations. Here, the fact that matrix multiplication is associative comes in handy as it allows to conduct those multiplications in a different order.

More precisely, assume, for sake of simplicity, that the number m of operations of a given building block or quantum algorithm is a power of two, i.e., $m = 2^k$ (for some $k \in \mathbb{N}$). Then, grouping the set of m operations into $m/2$ consecutive pairs, i.e.,

$$(U_{m-1} \cdot U_{m-2}) \cdot \ldots \cdot (U_3 \cdot U_2) \cdot (U_1 \cdot U_0) = U,$$

and performing the pairwise multiplications $(U_{i+1} \cdot U_i) = U_{i+1,i}$, leaves $m/2 = 2^{k-1}$ factors to be multiplied, i.e.,

$$U_{m-1,m-2} \cdot \ldots \cdot U_{1,0} = U.$$

Recursively applying this idea eventually results in the construction of the full functional representation $U \equiv U_{m-1,\ldots,0}$—requiring a total of k levels of pairwise grouping and multiplication. In case m is not a power of two, in some levels a pair may "degenerate" to a single operation.

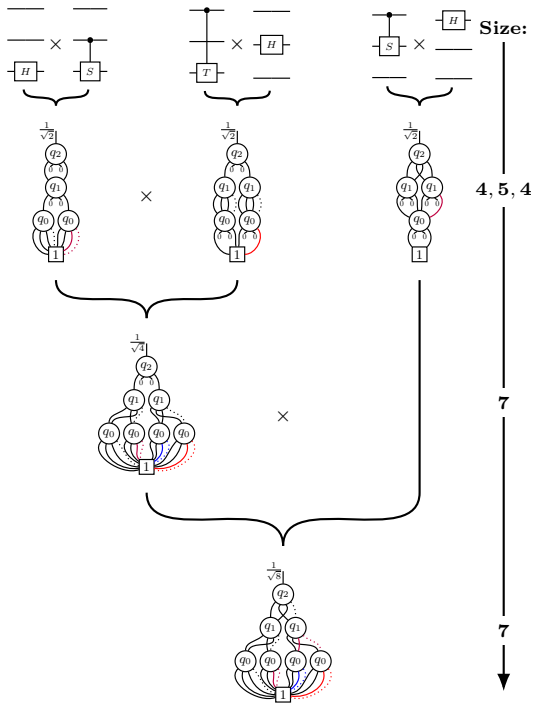


Fig. 4: Proposed approach applied to 3-qubit QFT

Example 5. Consider again the circuit for the 3-qubit QFT from Fig. 1d. Conducting the operations according to the proposed scheme results in the evolution of representations as sketched in Fig. 4. As can be seen, this leads to a much more efficient construction compared to the current state-of-the-art method illustrated before in Example 4: While, thus far, the multiplications resulted in intermediate representations with 4, 7, 7, 7, and 7 nodes (see Fig. 3), now the construction results in Decision Diagrams with 4, 5, and 4 nodes (first level), 7 nodes (second level), as well as 7 nodes (third level). While the total number of operations (as well as the final result) is obviously the same, more matrix-matrix multiplications are conducted on smaller representations. Furthermore, while, for small examples as considered here, this difference might seem negligible, evaluations on larger quantum algorithms show that this change in the order of multiplications has a substantial effect on the efficiency of the construction.

In general, employing the proposed scheme creates a tree-like hierarchy of matrix compositions. In each level $l \in \{0, \ldots, k\}$, at most 2^l operations contribute to a specific group of compositions. As a consequence, the intermediate functionalities during the construction can frequently be represented in a much more compact fashion (since these remain rather compact and/or sparse in many cases)—leading to fewer multiplications involving large representations.

Clearly, associativity of matrix multiplication allows for partitioning schemes beyond pairwise grouping. Determining an optimal partitioning scheme can be related to finding an optimal contraction order of a tensor network (itself an NP-hard problem [34]). In this sense, the proposed scheme can be viewed as one possible heuristic of tackling the contraction problem for quantum circuits.

At a first glance, the proposed scheme merely trades runtime for space: many operations can be conducted on rather small intermediate representations. But, this requires to store a lot more intermediate results when compared to sequential approaches—specifically in the first level of the multiplication hierarchy, where $m/2$ Decision Diagrams have to be stored. However, as those "early" intermediate results correspond to circuits with very low depth, their representations are rather compact and frequently contain redundant subparts that can be shared[8]. Moreover, the proposed approach can be realized using a stack for the intermediate results containing at most $\mathcal{O}(\log m)$ elements at any given time by proceeding in a depth-first fashion.

4.2 Exploiting Repeating Structures

Besides the general scheme proposed above, the made observations and findings can further be tailored to repeating structures in quantum algorithms—allowing for even more improvements in the construction of functional representations. This is described in the following section. To this end, recall that many quantum algorithms rely on repeated building blocks realizing a certain kind of iteration, e.g., Grover's search algorithm [35], Quantum Random Walks [36], Amplitude Estimation [29], or Phase Estimation [37]. Usually this type of algorithms consists of an initialization phase and an iteration phase comprised of multiple (identical) iteration steps. Inspired by emulation techniques [32], [33], the current state of the art accelerates the construction of the corresponding functional representation by constructing a single initialization matrix U_{init} followed by *multiple* multiplications with an iteration matrix U_{iter} (which has to be constructed only once), i.e., $(U_{iter} \cdot \ldots \cdot U_{iter})U_{init} = U$.

This procedure can be drastically improved further by, first, efficiently constructing the individual representations for U_{init} and U_{iter} using the general scheme proposed in Section 4.1 and, then, employing a binary exponentiation scheme for the sequence of multiplications involving U_{iter}. Assume, for sake of simplicity, that the number of iterations N is a power of two, i.e., $N = 2^k$ for some $k \in \mathbb{N}$. Then,

$$\overbrace{(U_{iter} \cdots U_{iter})}^{N} U_{init} = \overbrace{(U_{iter}^2 \cdots U_{iter}^2)}^{N/2} U_{init} = \ldots = U_{iter}^N U_{init} = U.$$

More precisely, once the iteration matrix U_{iter} has been efficiently constructed, it is sufficient to carry out only one multiplication $U_{iter}^{2^l} \cdot U_{iter}^{2^l} = U_{iter}^{2^{l+1}}$ (squar-

[8] Decision Diagram packages, e.g., typically employ a unique table where all nodes are stored [26]. Thus, even when multiple different Decision Diagrams are stored concurrently, sharing reduces the memory footprint considerably.

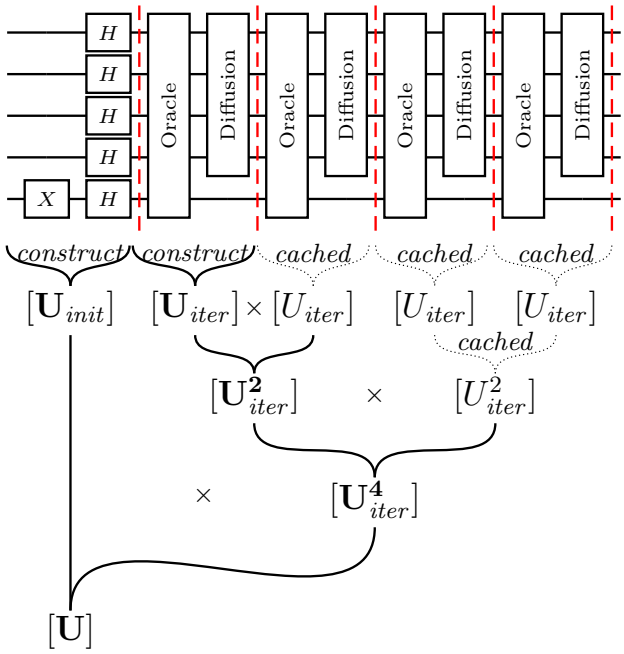


Fig. 5: Proposed strategy applied to Grover's algorithm

ing the current representation) at each level $l \in \{0, \ldots, k-1\}$, since all other multiplications are going to have the same result. As a last step, the initialization matrix U_{init} is multiplied to U^N_{iter}—yielding the desired representation U. Hence, only $k = \log_2(N) + 1$ (instead of N) building block multiplications are required to construct the representation—an exponential reduction. In case N is not a power of two, at most one additional multiplication per level is necessary. Thus, even in this case $\mathcal{O}(\log N)$ building block multiplications are sufficient for the construction. As a matter of fact, this does not just reduce the number of multiplications exponentially, it also avoids many computations on potentially large representations.

Example 6. In order to illustrate the idea we consider an application of Grover's algorithm [35]. The algorithm can be used to search for a specific item in an unstructured set of N items by only querying a given (problem-specific) oracle $\mathcal{O}(\sqrt{N})$ times—a quadratic speed-up over classical methods. To this end, it uses $\log(N) + 1$ qubits and consists of (1) a small initialization phase which puts all qubits into an equal superposition (to be represented by U_{init}) and (2) multiple Grover iterations (to be represented by U_{iter}). A single Grover iteration consists of querying a given oracle and, afterwards, applying the diffusion operator—effectively increasing the probabilities of states matching the search criterion encoded in the oracle.

Now, consider for example the case $N = 16$. This entails $\log(N)+1 = 5$ qubits and approximately $\sqrt{16} = 4$ Grover iterations. By first constructing the matrices U_{init} and U_{iter} (using the scheme described in Section 4.1) and, then, applying the approach proposed above, an evolution of representations as shown in Fig. 5 results. Here, it can be seen that, at each level, only a single multiplication has to be carried out (while the other multiplications are functionally equivalent and, hence, can be cached/reused). Thus, only three building block multiplications are required in total, while any sequential approach would need four multiplications. Again, this might look negligible for this small example, but has substantial effect once larger instances are considered.

5 Experimental Evaluations

In order to experimentally evaluate the proposed approaches, we implemented them on top of the publicly-available JKQ-framework [38] which includes the decision diagram package described in [26] and the state-of-the-art construction approach from [24] as reviewed in Section 3 and illustrated in Example 4. The resulting implementation has been integrated into the framework and is available at `https://github.com/iic-jku/qfr`. Afterwards, we used the resulting implementation to construct representations for the functionality of

- the Quantum Fourier Transform, as a representative for a common building block in quantum algorithms such as Shor's algorithm for integer factorization [28],
- Grover's search algorithm [35], as a representative of an algorithm containing repeated building blocks.

All computations have been performed on a machine with an Intel i7-6700K processor and 16 GiB RAM running macOS 11.2. The obtained results have been split into two parts and are shown in Table 1. In all tables, n and m denote the number of qubits and the number of gates, respectively. Furthermore, the runtime (in CPU seconds) as well as the total memory allocation (in GiB) needed to construct the respective representation is listed for

- the current state-of-the-art approach [24],
- the respective proposed techniques, i.e., the general scheme from Section 4.1 in Table 1a and the dedicated scheme for repeated structures from Section 4.2 in Table 1b.

The results for the QFT, which was used as a running example throughout this paper, clearly show that, compared to the current state of the art, the proposed method manages to construct the algorithm's functionality 3.0× faster on average (and up to 4.2× faster). On the one hand this shows that conducting as many operations as possible on as small as possible intermediate representations indeed pays off. On the other hand, it confirms the discussion from Section 4.1 that although the proposed technique requires to store more representations at the same time, possible redundancies/sharing can explicitly be exploited.

Table 1: Experimental results

(a) Results for the QFT

QFT		State of the art [24]		Prop. scheme 4.1	
n	m	t_{sota}	mem_{sota}	t_{prop}	mem_{prop}
12	84	0.03	0.07	**0.01**	0.07
13	97	0.03	0.07	**0.02**	0.07
14	112	0.05	0.08	**0.02**	0.07
15	127	0.15	0.09	**0.04**	0.08
16	144	0.37	0.09	**0.09**	0.09
17	161	1.01	0.10	**0.25**	0.10
18	180	3.79	0.11	**1.35**	0.12
19	199	10.05	0.16	**3.02**	0.18
20	220	14.72	0.20	**4.08**	0.23
21	241	21.01	0.23	**7.12**	0.29
22	264	27.04	0.27	**10.79**	0.33
23	287	33.42	0.31	**13.49**	0.39
24	312	39.83	0.34	**14.58**	0.39
25	337	46.48	0.38	**18.76**	0.43

(b) Results for Grover's algorithm

Grover		State of the art [24]		Prop. scheme 4.2	
n	m	t_{sota}	mem_{sota}	t_{prop}	mem_{prop}
12	1741	0.02	0.07	**0.01**	0.07
13	2614	0.02	0.07	**0.01**	0.07
14	3991	0.05	0.07	**0.01**	0.07
15	6076	0.09	0.08	**0.01**	0.07
16	9105	0.26	0.09	**0.02**	0.07
17	13686	0.34	0.08	**0.03**	0.07
18	20467	3.46	0.10	**0.03**	0.07
19	30572	3.84	0.10	**0.04**	0.07
20	45541	26.29	0.10	**0.05**	0.08
21	67558	68.50	0.10	**0.05**	0.08
22	100079	361.23	0.11	**0.07**	0.09
23	147960	>24.00 h	—	**0.08**	0.09
24	218425	>24.00 h	—	**0.12**	0.10
25	321726	>24.00 h	—	**0.15**	0.12

n: Number of qubits m: Number of gates t: Runtime in CPU seconds [s]
mem: Total memory allocations [GiB]

Drastic improvements can be achieved for quantum algorithms containing repeated structures for which the dedicated approach from Section 4.2 can be used. This is confirmed by the numbers provided in Table 1b: Here, the state-of-the-art method required 6 min to construct a representation for the Grover functionality for $n = 22$ and failed to construct the functionality at all within 24 h for larger instances. In contrast, the proposed approach managed to construct the functionality in *all* these cases within fractions of a second.

In a final series of evaluations, we aimed to compare the proposed techniques to IBM's toolchain Qiskit [13], specifically the CPU backend of the Qiskit Aer *UnitarySimulator* in version 0.7.1 which uses a multi-threaded array-based technique for constructing the functionality of a given circuit. The results for both the QFT as well as the Grover benchmarks are shown in Table 2. Even for moderately sized instances, we observed runtimes more than two orders of magnitude longer when compared to the technique from [24] or the techniques proposed in this paper. In addition, IBM's approach requires exponential amount of memory—leading to memory outs when considering more than 15 qubits while the proposed techniques easily allow to construct the functionality of circuits with more than 20 qubits.

Table 2: Comparison to IBM Qiskit [13]

QFT		IBM Qiskit		Grover		IBM Qiskit	
n	m	t	mem	n	m	t	mem
12	84	1.80	0.25	10	731	16.70	0.09
13	97	7.90	1.04	11	1 112	98.90	0.19
14	112	36.00	3.92	12	1 741	996.38	0.28
15	127	146.00	15.97	13	2 614	11 336.69	1.03
16	144	—	MemOut	14	3 991	>24.00 h	3.93
17	161	—	MemOut	15	6 076	>24.00 h	15.94
18	180	—	MemOut	16	9 105	—	MemOut
19	199	—	MemOut	17	13 686	—	MemOut

n: Number of qubits m: Number of gates
t: Runtime in CPU seconds [s] mem: Total memory allocations [GiB]

6 Conclusion

In this work, we addressed the issue of constructing the functional representation of certain building blocks or even entire quantum circuits. Existing approaches for solving this task are severely limited by the rapidly growing size of intermediate representations during the construction. By conducting as many operations as possible on as small as possible intermediate representations, the solutions proposed in this paper manage to consistently outperform existing approaches—allowing to construct the desired representations several factors faster than with the state of the art. Moreover, in case repeating structures are explicitly exploited, the construction of the representation for certain prominent quantum algorithms can be completed within seconds, whereas state-of-the-art approaches fail to construct it within an entire day. The comparison with IBM's Qiskit has shown that industrial tools for quantum computing are still in their infancy and would greatly benefit from the integration of existing techniques for efficiently constructing functional representations of quantum circuits—and even more so the techniques proposed in this work.

Acknowledgments This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 101001318). It has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria as well as by the BMK, BMDW, and the State of Upper Austria in the frame of the COMET program (managed by the FFG).

References

[1] P. Niemann, R. Wille, and R. Drechsler, "Improved synthesis of Clifford+T quantum functionality," *Design, Automation and Test in Europe*, pp. 597–600, 2018.

[2] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the IBM QX architectures," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1226–1236, 2019.

[3] A. Zulehner and R. Wille, "Compiling SU(4) quantum circuits to IBM QX architectures," in *Asia and South Pacific Design Automation Conf.*, Tokyo, Japan, 2019, pp. 185–190.

[4] T. Itoko, R. Raymond, T. Imamichi, A. Matsuo, and A. W. Cross, "Quantum circuit compilers using gate commutation rules," *Asia and South Pacific Design Automation Conf.*, pp. 191–196, 2019.

[5] K. N. Smith and M. A. Thornton, "Quantum logic synthesis with formal verification," *IEEE Int Midwest Symp Circuits Syst.*, pp. 73–76, 2019.

[6] A. Zulehner and R. Wille, "Advanced simulation of quantum computations," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 848–859, 2019.

[7] E. Pednault, J. A. Gunnels, G. Nannicini, L. Horesh, and R. Wisnieff. "Leveraging secondary storage to simulate deep 54-qubit Sycamore circuits." arXiv: `1910.09534`. (2019).

[8] B. Villalonga *et al.*, "A flexible high-performance simulator for verifying and benchmarking quantum circuits implemented on real hardware," *Npj Quantum Inf.*, vol. 5, no. 1, pp. 1–16, 2019.

[9] G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Checking equivalence of quantum circuits and states," in *Int'l Conf. on CAD*, 2007.

[10] S. Yamashita and I. L. Markov, "Fast equivalence-checking for quantum circuits," in *Int'l Symp. on Nanoscale Architectures*, 2010.

[11] L. Burgholzer, R. Raymond, and R. Wille. "Verifying results of the IBM Qiskit quantum circuit compilation flow." arXiv: `2009.02376 [quant-ph]`. (2020).

[12] L. Burgholzer and R. Wille, "Advanced equivalence checking for quantum circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2021.

[13] G. Aleksandrowicz *et al.*, "Qiskit: An open-source framework for quantum computing," *Zenodo*, 2019.

[14] *Cirq: A python framework for creating, editing, and invoking Noisy Intermediate Scale Quantum (NISQ) circuits.* [Online]. Available: `https://github.com/quantumlib/Cirq`.

[15] *Quantum Development Kit*, Microsoft. [Online]. Available: `https://microsoft.com/en-us/quantum/development-kit`.

[16] E. Gutiérrez, S. Romero, M. A. Trenas, and E. L. Zapata, "Quantum computer simulation using the CUDA programming model," *Computer Physics Communications*, vol. 181, no. 2, pp. 283–300, 2010.

[17] G. G. Guerreschi, J. Hogaboam, F. Baruffa, and N. P. D. Sawaya, "Intel Quantum Simulator: A cloud-ready high-performance simulator of quantum circuits," *Quantum Sci. Technol.*, vol. 5, p. 034 007, 2020.

[18] T. Jones, A. Brown, I. Bush, and S. C. Benjamin, "QuEST and high performance simulation of quantum computers," in *Scientific Reports*, 2018.

[19] V. Gheorghiu, "Quantum++: A modern C++ quantum computing library," *PLOS ONE*, vol. 13, no. 12, e0208073, 2018.

[20] I. L. Markov and Y. Shi, "Simulating quantum computation by contracting tensor networks," *SIAM J. Comput.*, vol. 38, no. 3, pp. 963–981, 2008.
[21] D. S. Wang, C. D. Hill, and L. C. L. Hollenberg, "Simulations of Shor's algorithm using matrix product states," *Quantum Inf Process*, vol. 16, no. 7, p. 176, 2017.
[22] J. D. Biamonte and V. Bergholm, "Tensor networks in a nutshell," 2017. arXiv: 1708.00006.
[23] A. Kissinger and J. van de Wetering, "PyZX: Large scale automated diagrammatic reasoning," presented at the Quantum Physics and Logic, vol. 318, 2019, pp. 229–241.
[24] P. Niemann, R. Wille, D. M. Miller, M. A. Thornton, and R. Drechsler, "QMDDs: Efficient quantum function representation and manipulation," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 86–99, 2016.
[25] S.-A. Wang, C.-Y. Lu, I.-M. Tsai, and S.-Y. Kuo, "An XQDD-based verification method for quantum circuits," in *IEICE Trans. Fundamentals*, 2008, pp. 584–594.
[26] A. Zulehner, S. Hillmich, and R. Wille, "How to efficiently handle complex values? Implementing decision diagrams for quantum computing," in *Int'l Conf. on CAD*, 2019.
[27] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.
[28] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, 1997.
[29] G. Brassard, P. Hø yer, M. Mosca, and A. Tapp, "Quantum amplitude amplification and estimation," in *Quantum computation and information*, ser. Contemp. Math. Vol. 305, 2002.
[30] A. Montanaro, "Quantum speedup of Monte Carlo methods," *Proc. of the Royal Society A*, vol. 471, 2015.
[31] P. Rebentrost, B. Gupt, and T. R. Bromley, "Quantum computational finance: Monte Carlo pricing of financial derivatives," *Phys. Rev. A*, vol. 98, 2018.
[32] D. S. Steiger, T. Häner, and M. Troyer, "ProjectQ: An open source software framework for quantum computing," *Quantum*, vol. 2, p. 49, 2018.
[33] A. Zulehner and R. Wille, "Matrix-Vector vs. Matrix-Matrix multiplication: Potential in DD-based simulation of quantum computations," in *Design, Automation and Test in Europe*, 2019.
[34] L. Chi-Chung, P. Sadayappan, and R. Wenger, "On optimizing a class of multi-dimensional loops with reduction for parallel execution," *Parallel Process. Lett.*, vol. 07, no. 02, pp. 157–168, 1997.
[35] L. K. Grover, "A fast quantum mechanical algorithm for database search," *Proc. of the ACM*, pp. 212–219, 1996.
[36] B. L. Douglas and J. B. Wang, "Efficient quantum circuit implementation of quantum walks," *Phys. Rev. A*, vol. 79, no. 5, p. 052 335, 2009.
[37] A. Y. Kitaev, "Quantum measurements and the abelian stabilizer problem," *Electron. Colloq. Comput. Complex.*, vol. 3, no. 3, 1996.
[38] R. Wille, S. Hillmich, and L. Burgholzer, "JKQ: JKU tools for quantum computing," in *Int'l Conf. on CAD*, 2020.