Advanced Simulation of Quantum Computations

Alwin Zulehner *Student Member, IEEE,* and Robert Wille *Senior Member, IEEE*
alwin.zulehner@jku.at robert.wille@jku.at

***Abstract*—Quantum computation is a promising emerging technology which, compared to conventional computation, allows for substantial speed-ups e.g. for integer factorization or database search. However, since physical realizations of quantum computers are in their infancy, a significant amount of research in this domain still relies on simulations of quantum computations on conventional machines. This causes a significant complexity which current state-of-the-art simulators try to tackle with a rather straight forward array-based representation and by applying massive hardware power. There also exist solutions based on decision diagrams (i.e. graph-based approaches) that try to tackle the exponential complexity by exploiting redundancies in quantum states and operations. However, these existing approaches do not fully exploit redundancies that are actually present.**

In this work, we revisit the basics of quantum computation, investigate how corresponding quantum states and quantum operations can be represented even more compactly, and, eventually, simulated in a more efficient fashion. This leads to a new graph-based simulation approach which outperforms state-of-the-art simulators (array-based as well as graph-based). Experimental evaluations show that the proposed solution is capable of simulating quantum computations for more qubits than before, and in significantly less run-time (several magnitudes faster compared to previously proposed simulators). An implementation of the proposed simulator is publicly available online at http://iic.jku.at/eda/research/quantum_simulation.

I. Introduction

Quantum computation [18] has become a promising technology which has theoretically been proven to be superior to conventional computation for important applications. For example, quantum algorithms for integer factorization (Shor's algorithm [28]) or database search (Grover's Search [10]) have been proposed that lead to significant – sometimes even exponential – speedups compared to conventional computations. With respect to physical implementations, significant progress has been made in the recent years as well [12], [26], [6], [17], [15]. The first publicly available quantum processor has been made accessible by IBM through their project *IBM Q* [2]. Via IBM's cloud infrastructure, the community can access a quantum processor with 5 qubits (launched in March 2017) and 16 qubits (launched in June 2017), respectively, to conduct experiments. IBM further plans to increase the number of available qubits to 50 – similar to Google's plans to provide a quantum chip with 49 qubits that demonstrates quantum supremacy [5], [8].

However, thus far, quantum computation remains an emerging technology. This requires, besides others, that respective developments have to be conducted while still relying on conventional technologies. In particular, this is an issue when it comes to simulating quantum computations or corresponding quantum algorithms. Although these quantum computations describe approaches to solve several problems significantly faster than a conventional technology, they still have to be simulated on conventional machines thus far. Furthermore, simulation plays an important role in the verification of existing and future quantum computers.

This causes a significant obstacle since basic and substantial concepts of quantum computations like superposition, entanglement, or measurement rely on exponentially large vector and matrix descriptions which additionally are composed of complex numbers. The majority of the existing methods for the simulation of quantum computations [9], [34], [29], [11], [14], [25], [30] address this problem using straight-forward methods like simple 1-dimensional and 2-dimensional arrays, respectively.[1] The resulting (exponential) complexity is then tackled by exploiting parallelism and applying massive hardware power such as supercomputers composed of thousands of nodes and more than a petabyte of distributed memory. But even then, quantum systems of rather limited size (today's practical limit is below 50 qubits [25]) can be simulated – additionally often requiring a significant amount of run-time (e.g. up to several days). Also current roadmaps show that also future plans rely on the use of massive hardware power, e.g. the authors of [29] expect to simulate 48-49 qubits on a machine with 4-10 petabytes of distributed memory.

In addition to that, simulators that utilize decision diagrams (e.g. [33], [27], [13]) have been proposed to tackle the exponential complexity of simulating quantum computations by exploiting redundancies. While decision diagrams have already been successfully used to solve exponential problems in the conventional domain (e.g. in verification [16] or synthesis [7]) significantly faster than straight-forward solutions, simulators for quantum computations based on decision diagrams (i.e. graph-based approaches) did not get established yet. In fact, the existing methods only exploit redundancies in a rather straight-forward fashion, which results in simulators that outperform array-based simulators for certain applications only.

In this work, we propose to use another type of decision diagram for the simulation of quantum computations that utilizes a decomposition scheme that is more natural to the occurring matrices and vectors – allowing to exploit even more redundancies. To this end, we revisit the basics of quantum computations and investigate how corresponding matrices and vectors can represented in a more efficient fashion. These endeavors eventually lead to a significantly more compact representation of quantum states and quantum operations than before, which exploits more redundancies in the corresponding description whenever possible. Furthermore, the natural decomposition scheme allows to realize dedicated manipulation

[1] Consequently, these methods are also called *array-based simulators*.

algorithms efficiently – leading to a new simulation method which clearly outperforms the current state-of-the-art.

In fact, the resulting compact representation allows for the simulation of well known quantum algorithms (such as Shor's Algorithm and Grover's Search) for more qubits than before. Finally, with respect to the run-time, a substantial drop can be observed: Instead of several days, the proposed approach is able to complete the simulations within hours – in many cases even just minutes or seconds. These improvements can be obtained with respect to array-based simulators as well as graph-based simulators.

This paper is structured as follows: Section II revisits the basics of quantum computation. In Section III, we investigate the obstacles of simulating quantum computations and review how the current state-of-the-art copes with these issues. Furthermore, the main idea of the utilized decision diagram is briefly sketched. In Sections IV and V we respectively discuss the resulting representation for vectors and matrices and the operations required to conduct the simulation in detail. In Section VI, we discuss the limitations of the approach and analyze the complexity of the required operations. Finally, the proposed solution is evaluated and compared to the state-of-the-art in Section VII, while Section VIII concludes the paper.

II. Quantum Computation

Quantum computation significantly differs from the conventional computation paradigm. To make this work self-contained and to properly introduce our solution, we first briefly revisit the basics on how operations are conducted in this domain. While this ought to be sufficient to comprehend the remainder of this paper, we refer to [18] for a more detailed treatment.

A. Quantum Bits

In conventional logic, information is represented by *bits* which can be in one of two basis states 0 and 1. Similarly, quantum computations rely on so called *quantum bits* (*qubits*) to represent internal states. Again, there exist two basis states, which – using the Dirac notation – are denoted $|0\rangle$ and $|1\rangle$. However, in contrast to bits in conventional logic, qubits are not restricted to one of these basis states, but may additionally assume an (almost) arbitrary superposition (i.e. a linear combination) of both. More precisely, the state of a qubit is described by $|\psi\rangle = \alpha_0 \cdot |0\rangle + \alpha_1 \cdot |1\rangle$, where the complex factors α_0 and α_1 denote *amplitudes* which indicate how much the qubit is related to the basis states.

The amplitudes of a quantum state $|\psi\rangle$ must satisfy the normalization constraint $|\alpha_0|^2 + |\alpha_1|^2 = 1$. While it is not possible to directly access the values of α_0 and α_1, it is possible to obtain one of the two basis states by measuring the qubit. More precisely, the basis state $|0\rangle$ is obtained with probability $|\alpha_0|^2$, while $|1\rangle$ is obtained with probability $|\alpha_1|^2$. The measurement collapses (i.e. destroys) the superposition. The concepts discussed above can be generalized for quantum systems composed of multiple qubits. Since each qubit has exactly two basis states, a system composed of n qubits has 2^n basis states – each one represented by $|\{0,1\}^n\rangle$. Overall, this accumulates in the following definition of a quantum state:

Definition 1. *Consider a quantum system composed of n qubits. Then, all possible states of the system are of form*

$$|\psi\rangle = \sum_{x\in\{0,1\}^n} \alpha_x \cdot |x\rangle\,,\ \textit{where} \sum_{x\in\{0,1\}^n} |\alpha_x|^2 = 1 \textit{ and } \alpha_x \in \mathbb{C}.$$

The state *$|\psi\rangle$ can be also represented by a column vector $\psi = [\psi_i]$ with $0 \le i < 2^n$ and $\psi_i = \alpha_x$, where $nat(x) = i$.*

Note that, to save space, vectors may be provided in their transposed form in the following (indicated by $[\cdot]^T$). That is, the single elements are listed horizontally rather than vertically.

Example 1. *Consider a quantum system composed of two qubits which is in the state $|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + 0 \cdot |01\rangle + 0 \cdot |10\rangle + \frac{1}{\sqrt{2}}|11\rangle$. This represents a valid state, since $\left(\frac{1}{\sqrt{2}}\right)^2 + 0^2 + 0^2 + \left(\frac{1}{\sqrt{2}}\right)^2 = 1$. The corresponding state vector is*

$$\psi = \left[\frac{1}{\sqrt{2}}, 0, 0, \frac{1}{\sqrt{2}}\right]^T.$$

Measuring this system yields one of the two basis states $|00\rangle$ or $|11\rangle$ – both with probability of $|\frac{1}{\sqrt{2}}|^2 = \frac{1}{2}$.

B. Quantum Operations

Quantum operations are used to manipulate the current state of a quantum system. All of them except the measurement are thereby inherently reversible and can be represented by unitary matrices U, i.e. a complex square matrix whose inverse is its conjugate transposed [18]. The size of the matrix depends on the number of involved qubits. Important quantum operations for a single qubit are e.g.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \text{and } Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},$$

where X complements the current state of the qubit, H sets the qubit into superposition, and Z changes the phase of the qubit, respectively. An important operation involving two qubits is e.g.

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

which performs a so-called controlled inversion. Here, one qubit serves as *control qubit*. The value the other qubit (i.e. the *target qubit*) is complemented if the *control qubit* is in base state $|1\rangle$. Consequently, the resulting matrix is composed of the 2×2 identity matrix in case that the *control qubit* is in basis state $|0\rangle$ (left upper quadrant) and the single qubit matrix X in case that the *control qubit* is in basis state $|1\rangle$. This concept can be easily extended to support single qubit gates that are controlled by multiple other qubits.[2]

[2] Note that the matrix grows exponentially with the number of control qubits.

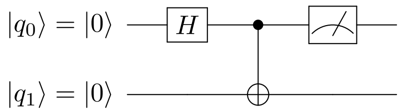


Fig. 1: Quantum circuit

To evaluate a quantum operation with respect to a given quantum state, the corresponding matrix U has to be multiplied with the corresponding state vector ψ. More precisely:

Definition 2. *Consider a quantum system composed of n qubits with*

- *a quantum operation U represented by a $2^n \times 2^n$ unitary matrix $U = [u_{i,j}]$ with $0 \leq i, j < 2^n$ and*
- *a system state $|\psi\rangle$ represented by a vector $\psi = [\psi_i]$ with $0 \leq i < 2^n$.*

Then, the output state $|\psi'\rangle$ of the quantum system is defined by a vector $\psi' = U \cdot \psi$, i.e. $\psi' = [\psi'_i]$ with

$$\psi'_i = \sum_{k=0}^{2^n-1} u_{i,k} \cdot \psi_k, \quad \textit{for } 0 \leq i < 2^n.$$

Example 2. *Consider a quantum system composed of two qubits which is currently in state $|\psi\rangle = |11\rangle$. Applying a* CNOT *operation yields*

$$\underbrace{\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}}_{CNOT} \cdot \underbrace{\begin{bmatrix}0\\0\\0\\1\end{bmatrix}}_{\psi} = \begin{bmatrix}0+0+0+0\\0+0+0+0\\0+0+0+1\\0+0+0+0\end{bmatrix} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \equiv |10\rangle .$$

Quantum circuits are used as proper description for a sequence of quantum operations. A *quantum circuit* [18] consists of a set of qubits, which are vertically aligned in a circuit diagram. The time axis is represented by a horizontal line for each qubit and read from left to right. Boxes on the time axis of a qubit indicate which quantum operation has to be applied. Note that measurement as reviewed in Section II-A and illustrated in Example 1 also counts as quantum operation in this context.

Example 3. *Consider the quantum circuit shown in Fig. 1. The circuit contains two qubits, q_0 and q_1, which are both initialized with basis state $|0\rangle$. Consequently, the initial state is $|\psi\rangle = |00\rangle$. First, a Hadamard operation is applied to qubit q_0, which is represented by a box labeled* H. *Then, a* CNOT *operation is performed, where q_0 is the* control qubit *(denoted by $\bullet$) and q_1 is the target qubit (denoted by $\oplus$). Finally, qubit q_0 is measured (represented by the meter symbol), which collapses its superposition into one of the two basis states.*

III. Conducting Simulation

The basics reviewed in the previous section are sufficient to simulate the execution of quantum operations. In fact, for a given sequence of quantum operations to be simulated, corresponding simulators simply have to conduct the multiplications of each operation matrix U with the respective intermediate quantum state $|\psi\rangle$ as reviewed in Def. 2 and illustrated in Example 2. However, for actual quantum algorithms severe challenges emerge which significantly restrict today's capabilities to simulate quantum computations. In the following, these challenges are discussed – followed by a summary of how state-of-the-art solutions currently deal with them.

A. Exponential Growth

A quantum circuit can be simulated by multiplying all matrices describing the quantum operation (from left to right) successively to the state vector. Therefore, all matrices have to be of dimension $2^n \times 2^n$. Since most quantum operations work on $k < n$ qubits only, their matrices have to be expanded to match the size of the state vector. To this end, an operation matrix for the remaining $n - k$ qubits is required. Since they shall not be affected by the gate, a 2×2 identity matrix I_2 is used for this purpose. The overall $2^n \times 2^n$-matrix is eventually obtained by forming the Kronecker product of all these matrices.

Example 4. *Consider again the quantum circuit shown in Fig. 1 with state $|q_0 q_1\rangle = |00\rangle$ as input. The first operation of the circuit is a Hadamard operation, which is applied to qubit q_0. Since this operation shall not affect q_1, we form the Kronecker product of* H *and the identity matrix I_2, i.e.*

$$H \otimes I_2 = \frac{1}{\sqrt{2}} \begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \otimes \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix}1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\\ 1 & 0 & -1 & 0\\ 0 & 1 & 0 & -1\end{bmatrix}.$$

Multiplying this matrix with the state vector yields

$$\psi' = \frac{1}{\sqrt{2}} \begin{bmatrix}1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\\ 1 & 0 & -1 & 0\\ 0 & 1 & 0 & -1\end{bmatrix} \cdot \begin{bmatrix}1\\0\\0\\0\end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix}1\\0\\1\\0\end{bmatrix}.$$

Applying the CNOT *operation yields*

$$\psi'' = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix} \cdot \frac{1}{\sqrt{2}} \begin{bmatrix}1\\0\\1\\0\end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix}1\\0\\0\\1\end{bmatrix}.$$

Since both, the state vectors as well as the operation matrices grow exponentially with respect to the number n of qubits, a crucial obstacle becomes evident: The simulation of quantum computations requires an exponential amount of space. The same complexity applies for the measurement of a quantum state, since, because of superposition, also the state vector may contain an exponentially large number of non-zero entries.

Now, one might think that a local consideration of qubits during the simulation avoids this exponential blow-up: Instead of forming a $2^n \times 2^n$-matrix using the Kronecker product, a simple application of an operation matrix to only those qubits which are actually affected might be sufficient. Unfortunately, this is not possible, since *entanglement*, which is one of the main concepts that make quantum computations superior to conventional computations, frequently occurs [18]. Two qubits are entangled if their state cannot be described without the other. An example illustrates the concept:

Example 4 (continued). *Consider again the quantum state $|\psi''\rangle$ from above. If we e.g. measure q_0, this qubit collapses to the basis state $|0\rangle$ or $|1\rangle$ with a probability $\left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}$. But due to the nature of $|\psi''\rangle$, this measurement also affects q_1. More precisely, if the measurement yields e.g. the basis state $|0\rangle$ for q_0, the new state vector is $\psi' = [1,0,0,0]^T$, i.e. also q_1 collapses to the basis state $|0\rangle$ (although not explicitly measured). This happens because $|\psi''\rangle$ represents a state in which both qubits are entangled.*

Since actual quantum computations frequently use entangled states, a local consideration of qubits affected by an operation is often not possible. Instead the complete (exponential) state vector is required.

B. State-of-the-Art Solutions

In the recent past, researchers and engineers intensely considered this problem and developed corresponding solutions for the simulation of quantum computations. Most of them are so-called *array-based approaches*, which are, however, rather limited, since they rely on a straight-forward representation of quantum states and operations (besides minor optimization, mainly representations such as simple 1-dimensional and 2-dimensional arrays are employed). Consequently, only experimental results for quantum systems with up to 34 qubits were reported on Desktop machines. In order to simulate quantum systems composed of more qubits, solutions exploiting massive hardware power (supercomputers composed of thousands of nodes and more than a petabyte of distributed memory) are applied. But even then, quantum systems with less than 50 qubits are today's practical limit [29], [25].

Besides that, solutions based on decision diagrams (so-called *graph-based simulators* [33], [27], [13]) have been proposed by researchers that try to exploit redundancies to gain a more compact representation of state vectors and matrices. One such type of decision diagrams suited for representing quantum computation are *Quantum Information Decision Diagrams* (QuIDDs [31]), which – like *Binary Decision Diagrams* (i.e. BDDs [4]) in conventional design – aim for a rather compact representation in many cases.

Overall, the current state-of-the-art approaches (*array-based* as well as *graph-based* ones) can be summarized as follows:

- *LIQUi$|\rangle$* [34]: Microsoft's tool suite for quantum computation with an integrated simulator which relies on a straight-forward representation and, thus, also can simulate systems with up to approximately 30 qubits only, when used on a Desktop machine with 32 GB RAM (still requires substantial run-times of up to several days).
- *qHiPSTER* [29]: A quantum high performance software testing environment developed in Intel's Parallel Computing Lab. Here, parallel algorithms are utilized which are executed on 1000 compute nodes with 32 TB RAM distributed across these nodes. Even with this massive hardware power, quantum systems of rather limited size (not more than 40 qubits) can be simulated.
- *Quantum Emulator proposed in* [11]: A solution which utilizes a higher level description of quantum computations (e.g. addition, quantum Fourier transformation, etc.) to directly compute intermediate results instead of applying the individual quantum operations successively. Experimental results are provided for systems with up to 36 qubits which, again, were accomplished with massive hardware power, i.e. a supercomputer similar to the one used for *qHiPSTER*.
- *QX* [14]: A high-performance array-based quantum computer simulation platform developed in the QuTech Computer Engineering Lab at Delft University. The simulator tries to parallelize the application of quantum gates to improve the performance. The authors state that *QX* allows for simulation of 34 fully entangled qubits on a single node using 270 GB of memory.
- *ProjectQ* [30]: ProjectQ is a software framework for quantum computing that started at the ETH Zurich. The contained high-performance array-based simulator allows to simulate up to approximately 30 qubits on a desktop machine. ProjectQ additionally contains an emulator, which can determine e.g. the result for Shor's algorithm significantly faster than the simulator by employing conventional shortcuts (e.g. arithmetic components are computed conventionally instead of using a quantum circuit).
- *QuIDDPro* [32], [33] is a graph-based simulator based on QuIDDs, which allows e.g. to simulate Grover's algorithm significantly faster than with array-based solutions by exploiting certain redundancies in the occurring state vectors and unitary matrices. However, the performance for simulating computations such as *Quantum Fourier Transformation* or *Shor's Algorithms* is rather limited since QuIDDs require a non-scalable number of decision diagram nodes for these cases (cf. Section VII).

C. General Idea

In this work, we investigate how quantum states and quantum operations can be represented more compactly so that an efficient simulation becomes possible. To this end, we utilize a decomposition scheme that is more natural to state vectors and unitary matrices used in simulation of quantum computations – allowing for a more compact representation and for developing efficient manipulation algorithms. In fact, some of these operations often boil down to rearranging pointers in the decision diagram. The general idea is motivated by the decomposition scheme of *Quantum Multiple-Valued Decision Diagrams* (QMDDs [23]), which have not yet been utilized in the context of simulating quantum computations. They rather have been applied to efficiently solve design tasks such as verification [35], [21] and synthesis [20], [19], [22], [36]. In the following sections, we introduce and discuss the proposed representations as well as the required manipulation algorithms in detail.

IV. Representations for Quantum Simulation

In this section, we discuss the proposed representation for state vectors and unitary matrices required for quantum simulation. To this end, we describe a compact representation for state vectors which, afterwards, is extended by a second dimension – leading to a compact representation for quantum operations.

A. Representation of State Vectors

As discussed in Section II-A, a system composed of n qubits is represented by a state vector of size 2^n – an exponential representation. However, a closer look at state vectors unveils that they are frequently composed of redundant entries which provide ground for a more compact representation.

Example 5. *Consider a quantum system with $n = 3$ qubits situated in a state given by the following vector:*

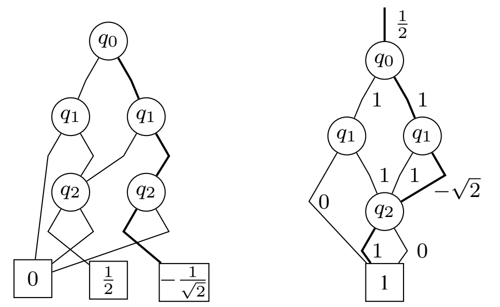
$$\psi = \left[0, 0, \frac{1}{2}, 0, \frac{1}{2}, 0, -\frac{1}{\sqrt{2}}, 0\right]^T .$$

Although of exponential size ($2^3 = 8$ entries), this vector only assumes three different values, namely 0, $\frac{1}{2}$, and $-\frac{1}{\sqrt{2}}$.

This redundancy can be exploited for a more compact representation. To this end, decision diagram techniques are employed. For conventional computations, e.g. *Binary Decision Diagrams* (BDDs, [4]) are very well known. Here, a decomposition scheme is employed which reduces a function to be represented into corresponding sub-functions. Since they also usually include redundancies, equivalent sub-functions result which can be shared – eventually yielding a much more compact representation. In a similar fashion, the concept of decomposition can also be applied to represent state vectors in a more compact fashion.

More precisely, similar to decomposing a function into sub-functions, we decompose a given state vector with its complex entries into sub-vectors. To this end, consider a quantum system with qubits $q_0, q_1, \ldots q_{n-1}$, whereby q_0 represents the most significant qubit.[3] Then, the first 2^{n-1} entries of the corresponding state vector represent the amplitudes for the basis states with q_0 set to $|0\rangle$; the other entries represent the amplitudes for states with q_0 set to $|1\rangle$. This decomposition is represented in a decision diagram structure by a node labeled q_0 and two successors leading to nodes representing the sub-vectors. The sub-vectors are recursively decomposed further until vectors of size 1 (i.e. a complex number) results. This eventually represents the amplitude α_i for the complete basis state and is given by a terminal node. During these decompositions, equivalent sub-vectors can be represented by the same nodes – allowing for sharing and, hence a reduction of the complexity of the representation. An example illustrates the idea.

Example 6. *Consider again the quantum state from Example 5. Applying the decompositions described above yields a decision diagram as shown in Fig. 2a. The left (right) outgoing edge of each node labeled q_i points to a node representing the sub-vector with all amplitudes for the basis states with q_i set to $|0\rangle$ ($|1\rangle$). Following a path from the root to the terminal yields the respective entry. For example, following the path highlighted bold in Fig. 2a provides the amplitude for the basis state with $q_0 = |1\rangle$ (right edge), $q_1 = |1\rangle$ (right edge), and $q_2 = |0\rangle$ (left edge), i.e. $-\frac{1}{\sqrt{2}}$ which is exactly the amplitude for basis state $|110\rangle$ (seventh entry in the vector from Example 5). Since some sub-vectors are equal (e.g. $\left[\frac{1}{2}, 0\right]^T$ represented by the left node labeled q_2), sharing is possible.*



(a) Without edge weights (b) With edge weights

Fig. 2: Representation of the state vector

However, even more sharing is possible. In fact, many entries of the state vectors differ in a common factor only (e.g. the state vector from Example 5 has entries $\frac{1}{2}$ and $-\frac{1}{\sqrt{2}}$ which differ by the factor $-\sqrt{2}$ only). This is additionally exploited in the proposed representation by denoting common factors of amplitudes as weights to the edges of the decision diagram. Then, the value of an amplitude for a basis state is determined by not only following the path from the root to the terminal, but additionally multiplying the weights of the edges along this path. Again, an example illustrates the idea.

Example 7. *Consider again the quantum state from Example 5 and the corresponding decision diagram shown in Fig. 2a. As can be seen, the sub-trees rooting the node labeled q_2 are structurally equivalent and only differ in their terminal values. Moreover, they represent sub-vectors $\left[\frac{1}{2}, 0\right]^T$ and $\left[-\frac{1}{\sqrt{2}}, 0\right]^T$ which only differ in a common factor.*

In the decision diagram shown in Fig 2b, both sub-trees are merged. This is possible since the corresponding value of the amplitudes is now determined not by the terminals, but the weights on the respective paths. As an example, consider again the path highlighted bold representing the amplitude for the basis state $|110\rangle$. Since this path includes the weights $\frac{1}{2}$, 1, $-\sqrt{2}$, and 1, an amplitude value of $\frac{1}{2} \cdot 1 \cdot (-\sqrt{2}) \cdot 1 = -\frac{1}{\sqrt{2}}$ results.

Note that, of course, various possibilities exist to factorize an amplitude. Hence, we apply a normalization which assumes the left edge to inherit a weight of 1. More precisely, the weights w_l and w_r of the left and right edge are both divided by w_l and this common factor is propagated upwards to the

[3] Note that, with the terminology *most-significant qubit*, we refer to a position in the basis states of a quantum system, rather than to the importance of the qubit itself.

parents of the node. If $w_l = 0$, the node is normalized by propagating w_r upwards to the parents of the node.[4]

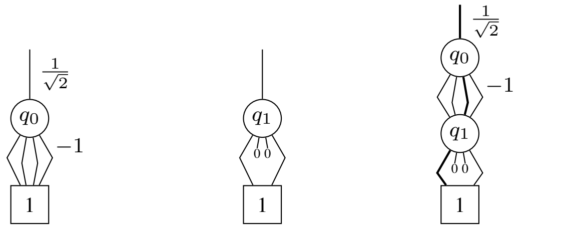
The resulting representation discussed above leads to the following definition.

Definition 3. *The resulting decision diagram for representing a 2^n-dimensional state vector is a directed acyclic graph with one terminal node labeled 1 that has no successors and represents a 1-dimensional vector with the element 1. All other nodes are labeled q_i, $0 \leq i < n$ (representing a partition over qubit q_i) and have two successors. Additionally, there is an edge pointing to the root node of the decision diagram. This edge is called* root edge*. Each edge of the graph has attached a complex number as weight. An entry of the state vector is then determined by the product of all edge weights along the path from the root towards the terminal. Without loss of generality, the nodes of the decision diagram are ordered by the significance of their label, i.e. the successor of a node labeled q_i are labeled with a less significant qubit q_j. The decision diagram is reduced, i.e. nodes where both outgoing edges point to the same successor and have attached the same weight (i.e. both sub-vectors are equal) are removed. Finally, the nodes are normalized, which means that all edges-weights are divided by the first non-zero weight. The common factor is propagated upwards in the decision diagram.*

B. Representation of Matrices

As discussed in Section II-B, quantum operations are described by unitary matrices. Similar to state vectors, these matrices include redundancies, which can be represented in a more compact fashion. To this end, we extend the proposed decomposition scheme for state vectors by a second dimension – yielding a decomposition scheme for $2^n \times 2^n$ matrices.

The entries of a unitary matrix $U = [u_{i,j}]$ indicate how much the operation U affects the mapping from a basis state $|i\rangle$ to a basis state $|j\rangle$. Considering again a quantum system with qubits $q_0, q_1, \ldots q_{n-1}$, whereby w.l.o.g. q_0 represents the most significant qubit, the matrix U is decomposed into four sub-matrices with dimension $2^{n-1} \times 2^{n-1}$: All entries in the left upper sub-matrix (right lower sub-matrix) provide the values describing the mapping from basis states $|i\rangle$ to $|j\rangle$ with both assuming $q_0 = |0\rangle$ ($q_0 = |1\rangle$). All entries in the right upper sub-matrix (left lower sub-matrix) provide the values describing the mapping from basis states $|i\rangle$ with $q_0 = |1\rangle$ to $|j\rangle$ with $q_0 = |0\rangle$ ($q_0 = |0\rangle$ to $q_0 = |1\rangle$). This decomposition is represented in a decision diagram structure by a node labeled q_0 and four successors leading to nodes representing the sub-matrices. The sub-matrices are recursively decomposed further until a 1×1 matrix (i.e. a complex number) results. This eventually represents the value $u_{i,j}$ for the corresponding mapping. Also during these decompositions, equivalent sub-matrices are represented by the same nodes and weights as well as a corresponding normalization scheme (as applied for the representation of state vectors) is employed.



(a) H (b) I_2 (c) $U = H \otimes I_2$

Fig. 3: Representation of matrices

Note that for a simpler graphical notation, we use zero stubs to indicate zero matrices (i.e. matrices that contain zeros only) and omit edge weights that are equal to one. Again, an example illustrates the idea.

Example 8. *Consider again the matrices of H, I_2, and $U = H \otimes I_2$ from Example 4. Fig. 3 shows the corresponding decision diagram representations. Following the path highlighted bold in Fig. 3c defines the entry $u_{0,2}$: a mapping from $|0\rangle$ to $|1\rangle$ for q_0 (third edge from the left) and from $|0\rangle$ to $|0\rangle$ for q_1 (first edge). Consequently the path describes the entry for a mapping from $|00\rangle$ to $|10\rangle$. Multiplying all factors on the path (including the* root edge*) yields $\frac{1}{\sqrt{2}} \cdot 1 \cdot 1 = \frac{1}{\sqrt{2}}$, which is the value of $u_{0,2}$.*

The concepts described above yield to the definition of a decision diagram representing a unitary matrix as follows.

Definition 4. *The resulting decision diagram for representing a $2^n \times 2^n$-dimensional unitary matrix is a directed acyclic graph with one terminal node labeled 1 that has no successors and represents a 1×1 matrix with the element 1. All other nodes are labeled q_i, $0 \leq i < n$ (representing a partition over qubit q_i) and have two successors. Additionally, there is an edge pointing to the root node of the decision diagram. This edge is called* root edge*. Each edge of the graph has attached a complex number as weight. An entry of the unitary matrix is then determined by the product of all edge weights along the path from the root towards the terminal. Without loss of generality, the nodes of the decision diagram are ordered by the significance of their label, i.e. the successor of a node labeled q_i are labeled with a less significant qubits q_j. The decision diagram is reduced, i.e. nodes where all outgoing edges point to the same successor and have attached the same weight (i.e. all four sub-matrices are equal) are removed. Finally, the nodes are normalized, which means that all edges-weights are divided by the first non-zero weight. The common factor is propagated upwards in the decision diagram.*

V. Conducting Quantum Simulations

With the availability of a compact representation for state vectors and unitary matrices, it is left to provide corresponding methods for conducting quantum operations, i.e. forming the Kronecker product, multiplying vectors with matrices, as well as measuring the quantum system. Since the applied decomposition scheme is natural to vectors and matrices, we can efficiently implement these required operations.

[4] Applying a fixed normalization scheme, a representation which is even canonic (w.r.t qubit order) results. However, since canonicity is not further relevant for the purpose of simulation, this issue is not discussed in detail in this work.

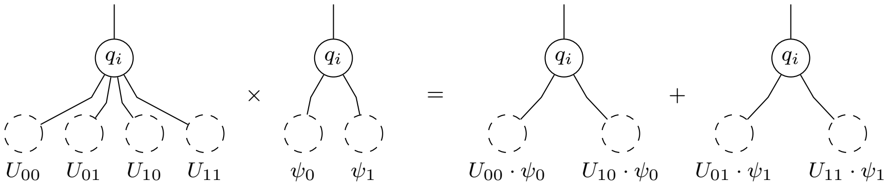


Fig. 4: Multiplication of a unitary matrix and a state-vector

A. Kronecker Product

As discussed in Section III, forming the Kronecker product of two matrices is essential to construct $2^n \times 2^n$ unitary matrices. Since we deal with square matrices whose dimensions are powers of 2 when simulating quantum computations, the Kronecker product is defined as

$$A \otimes B = \begin{bmatrix} a_{0,0}\cdot B & \cdots & a_{0,2^k-1}\cdot B \\ \vdots & \ddots & \vdots \\ a_{2^k-1,0}\cdot B & \cdots & a_{2^k-1,2^k-1}\cdot B \end{bmatrix}.$$

This means that each element $a_{i,j}$ of A has to be replaced by $a_{i,j}\cdot B$. While this constituted an expensive task using array-based realizations of A and B, it is very cheap to form the Kronecker product of two matrices given in the proposed decision diagram. This has also already been observed in [24].

Since $a_{i,j}$ is given as product of the edge weights from A's root node to the terminal and we can easily determine $a_{i,j}\cdot B$ by adjusting the weight of the edge pointing to B's root node. All that has to be done to determine $A \otimes B$ is replacing A's terminal with the root node of B. Additionally, the weight of A's root edge has to be multiplied by the weight of B's root edge.

Example 9. *Recall the matrices considered in Fig 3c. The Kronecker product $U = H \otimes I_2$ was efficiently constructed by taking the decision diagram representation of H (shown in Fig. 3a) and replacing its terminal node with the root node of the decision diagram representing I_2 (shown in Fig. 3b). Since the root edge of I_2 has weight 1, the value of the root node of U is equal to the weight of A's root edge.*

Note that forming the Kronecker product is not that simple when using other types of decision diagrams like QuIDDs. Here, the complex entries of the matrices are stored in different terminals rather than in edge weights. Consequently, forming the Kronecker product of two matrices A and B represented by QuIDDs requires – among others – to multiply all terminals of A with those of B.

B. Multiplying Unitary Matrices

The vector/matrix-multiplication as defined in Def. 2 can also be decomposed with respect to the most significant qubit leading to

$$\psi'_i = \sum_{k=0}^{2^n-1} u_{i,k}\cdot\psi_k = \sum_{k=0}^{2^{n-1}-1} u_{i,k}\cdot\psi_k + \sum_{k=2^{n-1}}^{2^n-1} u_{i,k}\cdot\psi_k,$$

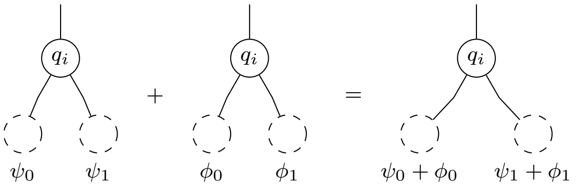


Fig. 5: Addition of state-vectors

or, using the matrix notation,

$$U\cdot\psi = \begin{bmatrix} U_{00} & U_{01} \\ U_{10} & U_{11} \end{bmatrix} \cdot \begin{bmatrix} \psi_0 \\ \psi_1 \end{bmatrix} = \begin{bmatrix} U_{00}\cdot\psi_0 \\ U_{10}\cdot\psi_0 \end{bmatrix} + \begin{bmatrix} U_{01}\cdot\psi_1 \\ U_{11}\cdot\psi_1 \end{bmatrix}.$$

This means, that we have to recursively determine[5] the four sub-products $U_{00}\cdot\psi_0$, $U_{01}\cdot\psi_1$, $U_{10}\cdot\psi_0$, and $U_{11}\cdot\psi_1$. As shown in Fig. 4, these sub-products are then combined with a decision diagram node to two intermediate state vectors. Finally, these intermediate state vectors have to be added. This addition can be recursively decomposed in a similar fashion, namely

$$\psi + \phi = \begin{bmatrix} \psi_0 \\ \psi_1 \end{bmatrix} + \begin{bmatrix} \phi_0 \\ \phi_1 \end{bmatrix} = \begin{bmatrix} \psi_0+\phi_0 \\ \psi_1+\phi_1 \end{bmatrix}.$$

The recursively determined sub-sums $\psi_0+\phi_0$ and $\psi_1+\phi_1$ are composed by a decision diagram node as shown in Fig. 5.

Moreover, all these decompositions into sub-products and sub-sums do not change the decision diagram structure. Hence, the complexity of them remains bounded by the number of nodes of the original representations. Furthermore, redundancies can again be exploited by caching sub-products and sub-sums.

C. Measurement

Measurement can also be conducted efficiently on the decision diagram structure. To this end, consider w.l.o.g. that qubit q_0 (which is represented by the root node of the corresponding decision diagram) of the state vector should be measured (this can easily be accomplished by applying a SWAP operation or by re-arranging the nodes and edges of the decision diagram). Then, the left (right) successor of the root node represents the sub-vector containing the amplitudes

[5] The decompositions of multiplication and addition are recursively applied until 1×1 matrices or 1-dimensional vectors result. Since those eventually represent just complex numbers, their multiplication and/or addition is straight-forward.

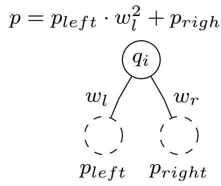


Fig. 6: Probability of a decision diagram node

of all states with $q_0 = |0\rangle$ ($q_0 = |1\rangle$), i.e. states that are of form $|0q_1q_2\ldots\rangle$ ($|1q_1q_2\ldots\rangle$). The probability for collapsing qubit q_0 to one of the two basis states is defined as follows.

Definition 5. *Consider a quantum system composed of n qubits $q_0, q_1, \cdots q_{n-1}$. Then, the probability measuring (and, thus, collapsing to) basis state $|0\rangle$ (basis state $|1\rangle$) for qubit q_0 is the sum of the squared magnitudes of the complex entries in the corresponding sub-vector, i.e.*

$$P(q_0 \to |0\rangle) = \sum_{x \in 0\{0,1\}^{n-1}} |\alpha_x|^2$$
$$P(q_0 \to |1\rangle) = \sum_{x \in 1\{0,1\}^{n-1}} |\alpha_x|^2 .$$

Example 10. *Consider again the quantum state discussed in Example 5. The probabilities for measuring $q_0 = |0\rangle$ and $q_0 = |1\rangle$ are:*

$$P(q_0 \to |0\rangle) = |0|^2 + |0|^2 + \left|\frac{1}{2}\right|^2 + |0|^2 \qquad = \frac{1}{4}$$
$$P(q_0 \to |1\rangle) = \left|\frac{1}{2}\right|^2 + |0|^2 + \left|-\frac{1}{\sqrt{2}}\right|^2 + |0|^2 = \frac{3}{4}$$

Therefore, the qubit q_0 is collapsed into basis state $|0\rangle$ (basis state $|1\rangle$) with a probability of 0.25 (0.75).

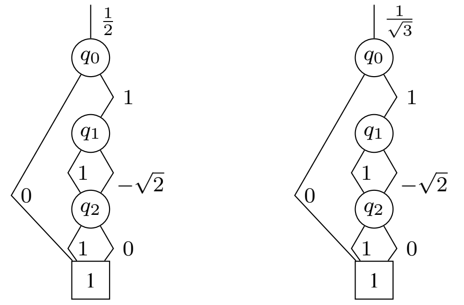
Consequently, we have to determine the summed probabilities for the decision diagram nodes. Again, the calculation of these probabilities can recursively be decomposed since

$$\sum_{x \in 0\{0,1\}^{n-1}} |\alpha_x|^2 = \sum_{x \in 00\{0,1\}^{n-2}} |\alpha_x|^2 + \sum_{x \in 01\{0,1\}^{n-2}} |\alpha_x|^2 .$$

This means we have to recursively determine the summed probabilities p_{left} and p_{right} of the sub-vectors. As Fig. 6 shows, the summed probability of the current decision diagram node is then determined by the sum of the probabilities of the sub-vectors. Before these probabilities are added, they are multiplied with the squared weight of the respective edges.

Note that, in the proposed decision diagram representation, the amplitudes α_x of the 2^n basis states are determined by a product of $n+1$ edge weights, i.e. $\alpha_x = \prod_{i=0}^{n} w_{x,i}$. Since $|\alpha \cdot \beta| = |\alpha| \cdot |\beta|$ holds for all complex numbers $\alpha, \beta \in \mathbb{C}$, $|\alpha_x|^2$ can be determined on the decision diagram, i.e.

$$|\alpha_x|^2 = \left|\prod_{i=0}^{n} w_{x,i}\right|^2 = \prod_{i=0}^{n} |w_{x,i}|^2 .$$



(a) Measure $q_0 = |1\rangle$ (b) Normalize amplitudes

Fig. 7: Measurement of qubit q_0

Finally, the weight on the edges to the left and the right successor of the root node (as well as the weight of the root edge) have to be considered to obtain the correct probabilities $P(q_0 \to |0\rangle)$ and $P(q_0 \to |1\rangle)$. An example illustrates the idea.

Example 10. *The decision diagram shown in Fig 2b represents the quantum state ψ (cf. Example 5). The probability of the node labeled q_2 can be determined by $1^2 + 0^2 = 1$. Based on that, the probabilities of the two nodes labeled q_1 can be determined. These are $0^2 \cdot 1 + 1^2 \cdot 1 = 1$ for the left node and $1^1 \cdot 1 + \left|-\sqrt{2}\right|^2 \cdot 1 = 3$ for the right node. From these nodes, we can determine the probabilities for collapsing q_0 to basis state $|0\rangle$ or $|1\rangle$ by*

$$\begin{aligned} P(q_0 \to |0\rangle) &= \left(\frac{1}{2}\right)^2 \cdot 1^2 \cdot 1 &= \frac{1}{4} \\ P(q_0 \to |1\rangle) &= \left(\frac{1}{2}\right)^2 \cdot 1^2 \cdot 3 &= \frac{3}{4} \end{aligned} .$$

Having the probabilities for collapsing q_0 to basis state $|0\rangle$ and $|1\rangle$ allows to sample the new value for q_0. If we obtain basis state $|0\rangle$ (basis $|1\rangle$), the amplitudes for all basis states with $q_0 = |1\rangle$ ($q_0 = |0\rangle$) drop to zero. In the decision diagram, we perform this collapse by changing the right (left) outgoing edge of the root node to point to the terminal and attach weight zero. Finally, the remaining (non-zero) amplitudes in the state vector must be modified in order to fulfill the normalization constraint (cf. Section II-A). To this end, all amplitudes are divided by $\sqrt{P(q_0 \to |0\rangle)}$ ($\sqrt{P(q_0 \to |1\rangle)}$). This can easily be conducted on the decision diagram structure by modifying the weight of the root edge.

Example 10 (continued)**.** *Assume we measure basis state $|1\rangle$ for qubit q_0. Fig. 7a shows the resulting decision diagram. To fulfill the normalization constraint, we divide the weight of the edge to the root node by $\sqrt{\frac{3}{4}}$ – eventually resulting in the decision diagram shown in Fig. 7b.*

Measuring all qubits can be conducted in a similar fashion. In fact, we repeat the procedure discussed above sequentially for all qubits $q_0, q_1, \cdots q_{n-1}$. Assume that qubit q_i shall be measured, and that all qubits q_j where $j < i$ are already measured. Then, there exists only one node labeled q_i, which is the root node of the sub-vector to be measured.

VI. Discussion

In this section, we discuss the complexity of the proposed approach with respect to existing array-based and graph-based solutions. This allows to get an intuition why the proposed approach can significantly outperform them in many cases. We distinguish thereby between a discussion about the representation of vectors and matrices as well as a discussion about the respective operations.

A. Representation of Vectors and Matrices

While array-based approaches always have to deal with 2^n-dimensional state vectors and $2^n \times 2^n$-dimensional matrices, graph-based approaches often allow for a significantly more compact representation in many practically relevant cases. This is similar to BDDs in conventional design, which have an exponential size for most Boolean functions (especially random ones), but allow rather compact representation for many functions of interest. As already mentioned in Section IV-A, introducing edge weights and a normalization scheme allows to exploit more redundancies than previous graph-based solutions – leading to an even more compact representation.

Nevertheless, in the worst case (i.e. if no redundancies in the state vector can be exploited), a full binary tree with $|v| = 1 + \sum_{i=0}^{n-1} 2^i = 2^n$ nodes results. Furthermore, $2 \cdot (2^n - 1) + 1 = 2^{n+1} - 1$ complex edge weights have to be stored – approximately twice as many complex numbers than used in array-based solutions and when using e.g. QuIDDs to represent the state-vector. That is, in the absolute worst case, the proposed representation is twice as large as array-based and graph-based solutions. This additional overhead, however, allows to exploit much more redundancies and, hence, to eventually gain more compact representations. This is confirmed by our experimental evaluation, which clearly shows that the peak node count of the proposed approach is significantly below that exponential upper bound.

In general, the worst-case memory complexity for $2^n \times 2^n$ matrices is analogous to the memory complexity for state-vectors – a full quad-tree has $|v| = 1 + \sum_{i=0}^{n-1} 4^i = 1 + \frac{4^n-1}{3}$ nodes. However, elementary quantum operations considered in this work (cf. Section II-B) that are composed of a single target qubit and an arbitrary number of control qubits only require a linear (in the number of qubits) number of nodes (cf. Section V-A).

B. Conducting Operations

Matrix operations are also less complex for graph-based solutions compared to array-based ones, which always suffer from an exponential complexity (since the exponentional matrix and the vector have to be traversed several times). However, there are also differences between the individual graph-based solutions.

As discussed in Section V-A, the Kronecker product of two matrices can easily be determined on the proposed type of decision diagrams by exchanging the terminal of one matrix with the root node of the other matrix. Consequently, forming the Kronecker product has complexity of $O(|v|)$. In contrast, this is more complex when using e.g. QuIDDs, where it is required – among others – to multiply all terminal values of the two matrices with each other.

Also matrix vector multiplication can be performed significantly faster for graph-based solutions. The complexity of a matrix multiplication has an upper bound defined by the product of the number of nodes needed to represent the state-vector and the number of nodes needed to represent the matrix. Since we only consider matrices where the number of nodes grows linearly with the number of qubits, the overall complexity is $O(n \cdot |v|)$.

Also measuring all qubits of a state vector has complexity $O(|v|)$. This is, because each node has to be traversed only once. Afterwards, each qubit can be measured in $O(1)$, starting at the top of the decision diagram – resulting in an overall complexity of $O(|v| + n)$.

Overall, this clearly shows that graph-based solutions offer more efficient representation and manipulation of state vectors in many cases. Although graph-based approaches suffer from overhead caused by the decision diagram structure (and additional complex numbers), these approaches can outperform array-based solutions by exploiting redundancies. Since the proposed graph-based approach exploits even more redundancies than e.g. QuIDDPro, we also observe a significant performance improvement compared to this representative. This is confirmed by our empirical evaluation summarized in the next section.

VII. Experimental Results

We evaluated the scalability of the proposed approach and compared it to the state-of-the-art. To this end, we implemented the simulator in C++[6] on top of the QMDD package provided by [23], which we extended and modified to realize the concepts introduced above. As state-of-the-art, we considered the publicly available implementations of the recently proposed array-based simulators *LIQUi*$|\rangle$ [34], *QX* [14] and the simulator of *ProjectQ* [30][7], as well as the graph-based simulator *QuIDDPro* [33]. All simulations have been conducted on a regular Desktop computer, i.e. a 64-bit machine with 4 cores (8 threads) running at a clock frequency of 3.8 GHz and 32 GB of memory running Linux 4.4.[8] Besides that, we additionally considered the best results published for other simulators (cf. Section III-B) that have been taken from the respective papers.

As benchmarks, well-known quantum algorithms considered by previous work have been used. More precisely, quantum systems generating entangled states, conducting *Quantum Fourier Transformation* (QFT; cf. [18]), executing Grover's Algorithm for database search [10], and executing Shor's Factorization Algorithm [28] (using the realization

[6] The implementation is publicly available at http://iic.jku.at/eda/research/quantum_simulation

[7] Note that ProjectQ also provides an emulator, where high level operations are applied directly. However, in order to allow for a fair evaluation, we only compared the simulators against each other and not a simulator against an emulator.

[8] The proposed approach as well as QuIDDPro use a single core while the simulators LIQ$Ui|\rangle$, QX, and the simulator of ProjectQ use multiple threads.

proposed by Beauregard [3] that requires $2n + 3$ qubits to factor an n-bit integer) have been considered. Note that, for all benchmarks except QFT, the initial assignments of the inputs are fixed. For QFT, we randomly chose one of the basis states as initial input assignment.

In order to not run into numerical issues when normalizing the decision diagrams (which requires many divisions), we used the *GNU MPFR* library [1] to increase the precision of the floating point numbers and checked at each measurement whether the probabilities for measuring one of the basis states sum up to 1 (except a tiny ϵ). In fact, using a precision of 200 bits was enough the avoid numerical errors for all considered benchmarks.

Table I summarizes the results. The columns *#Qubits* and *#Ops* list the number of involved qubits and the number of quantum operations to be conducted, respectively. In the remaining columns, we list the simulation time (i.e. the entire runtime from initialization to termination) and peak memory when using LIQ$Ui|\rangle$, QX, ProjectQ, QuIDDPro, as well as the proposed approach. For the graph-based simulators (i.e. QuIDDPro as well as the proposed one), we also list the peak node count during simulation, which gives a more accurate measurement of scalability than the actual memory consumption. Besides that, Table I also provides a comparison to other state-of-the-art simulation approaches. That is, whenever results from them are provided in the literature, the respectively best result for a considered quantum algorithm is summarized in the bottom of that part.

Note that some restrictions apply for certain state-of-the-art simulators: In fact, the publicly available version of LIQ$Ui|\rangle$ allows to simulate circuits composed of at most 23 qubits only. Using QX, we were able to simulate up to 29 qubits on our machine – trying to allocate 30 or more qubits failed (due to limited memory). Furthermore, QX does not allow to simulate Beauregard's realization of Shor's algorithm for integer factorization, because of missing features in the circuit description language (since QX is still in its infancy). We have accordingly marked all these cases by *n.a.* (not applicable) in Table I. Furthermore, the current release of QuIDDPro (version 3.8) also contains an improved simulator called *QuIDDProLite* (to be activated with the command line option *-cs*) that runs on average four times faster than QuIDDPro. However, this improved version can only simulate stand-alone quantum circuits and, hence, is not applicable for simulating Beauregard's implementation of Shor's algorithm (which also requires non-quantum control structures). Consequently, we list the runtime of QuIDDPro for Shor's algorithm and the runtime of QuIDDProLite for the other benchmarks (i.e. always the best result of both QuIDDPro versions are reported). Since QuIDDProLite does not offer the capability to dump the peak node count, we list the number of nodes obtained when using QuIDDPro for simulation. In the cases where this simulation does not succeed within the given timeout of five hours, we again label the corresponding entry by *n.a.* (not applicable).

As can be seen, the simulation of quantum systems generating entangled states and conducting QFT shows a linear behavior on our simulator. While this allows for a rather unlimited scalability using the solution proposed in this work, Microsoft's simulator LIQ$Ui|\rangle$, QX, as well as ProjectQ show exponential behavior. Even massive hardware power such as employed by Intel's simulator *qHiPSTER* [29] (running on a machine with 1000 nodes and 32 terabytes of memory) or the quantum emulator of [11] (running on a similar machine) manages to conduct QFT for a maximum of 40 qubits only (and additionally requires hundreds of seconds, while the approach proposed in this work terminates in a fraction of a second).

The graph-based simulator QuIDDPro [33] is capable to efficiently conduct entanglements (similar to the proposed approach, only a linear amount of nodes is required). But also here, an exponential behavior can be observed when conducting QFT. This is caused by the fact that the state vector contains exponentially many different entries which cannot be handled efficiently by QuIDDPro. In contrast, the proposed solution can exploit further redundancies here (namely subvectors which are multiples of each other as discussed in Section IV) – resulting in a linear number of decision diagram nodes for QFT.

The simulation of Grover's Algorithm and Shor's Algorithm constitutes a more challenging task. But even here, the proposed representation remains rather compact. For example, in case of simulating Shor's Algorithm with 37 qubits, only slightly more than 20 000 nodes are required. In fact, the significantly larger number of operations is more challenging here. Nevertheless, the proposed approach still manages to simulate both algorithms significantly more efficient and for more qubits than the state-of-the-art. While e.g. Microsoft's simulator *LIQ$Ui|\rangle$* is capable of conducting Shor's Algorithm for at most 31 qubits in more than 30 days (on a similar machine; cf. [34]), the simulation approach proposed in this work completes this task within in less than a minute.

Also the previously proposed graph-based solution QuIDDPro can not reach this efficiency. While QuIDDPro allows for a rather compact representation when simulating Grover's Algorithm, it requires a substantial amount of runtime. This is caused by the fact that the required operations cannot be conducted as efficiently as with the proposed solution (since the decomposition scheme is not that natural to matrices and state-vectors as discussed in Section VI). Hence, the proposed solution clearly outperforms QuIDDPro (which is optimized for Grover's Algorithm and has mainly been evaluated on that in the literature). For Shor's Algorithm, we can observe similar limitations for QuIDDPro than for array-based solutions (in fact, LIQ$Ui|\rangle$ is performing even better than QuIDDPro in this case). Again, the solution proposed in this work can handle all these cases much faster and for more qubits than before.

Overall, the proposed simulation approach clearly outperforms the current state-of-the-art in terms of runtime as well as in terms of memory (only up to 260 MB were required). Additionally, the proposed solution is able to simulate quantum computations for more qubits. Besides that, all these accomplishments can be achieved on a single core of a regular Desktop machine, i.e. without massive hardware power or the utilization of supercomputers.

TABLE I: Experimental results

			LIQ$Ui|\rangle$ [34]		QX [14]		ProjectQ [30]		QuIDDPro [33]			Proposed approach		
Computation	#Qubits	#Ops	Time [s]	Mem [MB]	Time [s]	Mem [MB]	Time [s]	Mem [MB]	Time [s]	Mem [MB]	#Nodes	Time [s]	Mem [MB]	#Nodes
Entanglement	22	22	3.53	193.33	0.42	200.47	1.08	152.41	0.04	14.93	45	<0.01	48.02	43
	23	23	4.09	248.25	0.80	396.94	0.49	248.01	0.04	14.92	47	<0.01	48.03	45
	24	24	n.a.		1.61	790.23	0.64	444.69	0.04	14.93	49	<0.01	48.00	47
	26	26	n.a.		6.82	3149.66	2.74	1624.56	0.04	14.93	53	<0.01	48.02	51
	28	28	n.a.		30.53	12586.87	4.91	6342.95	0.04	14.93	57	<0.01	48.14	55
	29	29	n.a.		63.02	25169.79	9.12	12634.67	0.04	14.92	59	<0.01	48.24	57
	30	30	n.a.		n.a.		17.76	25217.66	0.04	14.93	61	<0.01	48.14	59
	31	31	n.a.		n.a.		MO		0.04	14.93	63	<0.01	48.11	61
	100	100	n.a.		n.a.		MO		0.14	15.98	201	<0.01	49.32	199
	• Reported for QX [14]: max. 34 qubits using less than 270 GB of memory													
QFT	18	171	3.02	192.91	0.27	16.56	0.87	57.82	24.21	192.77	65535	0.01	48.47	18
	20	210	4.35	188.67	1.59	53.49	0.50	75.82	2263.38	1210.34	2097151	0.01	48.56	20
	21	231	6.46	192.83	3.63	102.75	0.66	100.75	10208.06	2511.26	4194303	0.01	48.78	21
	22	253	11.38	191.00	8.06	201.01	1.06	150.30	MO			0.01	48.92	22
	23	276	22.43	312.41	15.06	397.50	1.55	250.18	MO			0.01	49.13	23
	24	300	n.a.		31.66	790.77	3.27	445.04	MO			0.01	49.19	24
	26	351	n.a.		139.23	3150.14	12.84	1624.81	MO			0.02	49.36	26
	29	435	n.a.		1270.03	25170.21	109.19	12638.54	MO			0.02	49.95	29
	30	465	n.a.		n.a.	25217.39	234.27		MO			0.02	50.00	30
	31	496	n.a.		n.a.		MO		MO			0.03	50.53	31
	64	2080	n.a.		n.a.		MO		MO			0.09	68.00	64
	• Reported for qHiPSTER (Intel, cf. [29]): max. 40 qubits on a supercomputer (in approx. 1000 s). • Reported for Quantum emulator from [11]: max. 36 qubits on a supercomputer (in approx. 10 s).													
Grover	16	11600	97.78	195.11	55.90	57.28	6.65	51.88	6.93	16.29	39	0.14	50.54	130
	18	26082	770.26	193.42	583.45	144.41	16.37	58.12	23.49	17.24	44	0.33	50.78	148
	20	57940	8494.59	198.22	6394.54	382.51	77.95	77.17	85.86	19.05	39	0.78	50.80	166
	21	86037	>18000.00		>18000.00		229.45	101.27	168.07	20.60	n.a.	0.97	50.91	175
	24	278040	n.a.		>18000.00		5362.00	447.11	1272.36	31.78	n.a.	3.11	51.14	202
	25	409625	n.a.		>18000.00		15765.00	843.85	2598.08	40.55	n.a.	5.55	51.14	211
	26	602394	n.a.		>18000.00		>18000.00		5076.64	52.95	n.a.	8.24	51.26	220
	27	884763	n.a.		>18000.00		>18000.00		>18000.00			14.65	51.26	238
	30	2780430	n.a.		n.a.		>18000.00		>18000.00			37.23	51.32	256
	40	118632840	n.a.		n.a.		>18000.00		>18000.00			1239.96	52.01	346
Shor	13	12640	76.75	65.88	n.a.		0.40	47.56	1665.28	92.03	16375	0.21	52.65	40
	15	23083	298.59	62.72	n.a.		9.30	51.43	16236.14	365.22	65535	0.54	55.09	72
	17	38847	343.83	128.81	n.a.		19.25	55.10	>18000.00			0.76	57.63	66
	19	61510	1232.83	85.01	n.a.		47.48	70.91	>18000.00			1.07	60.56	156
	21	92700	7888.00	147.03	n.a.		187.14	115.74	>18000.00			10.98	64.88	519
	23	134490	>18000.00		n.a.		947.72	283.88	>18000.00			3.35	67.30	151
	25	188784	n.a.		n.a.		4827.00	860.96	>18000.00			17.91	75.17	653
	27	258060	n.a.		n.a.		>18000.00		>18000.00			74.52	83.34	949
	31	451577	n.a.		n.a.		MO		>18000.00			44.60	97.91	305
	33	581272	n.a.		n.a.		MO		>18000.00			1019.55	156.12	6517
	37	922385	n.a.		n.a.		MO		>18000.00			5585.64	259.96	20917
	• Reported for LIQ$Ui|\rangle$ (Microsoft, cf. [34]): max. 31 qubits (in more than 30 days).													

Time denotes the actual runtime and not the CPU seconds (which would be even higher for [34], [14], [30] since they use parallelization to speed-up simulation).

VIII. Conclusions

This work introduces a new graph-based approach for the simulation of quantum computations that clearly outperform previous graph-based or array-based solutions. To this end, we revisited the basics of quantum computation and developed a simulation approach which exploits redundancies in the respective quantum state and operation descriptions. The resulting simulator (which is publicly available at http://iic.jku.at/eda/research/quantum_simulation) is capable of (1) simulating quantum computations for more qubits than before, (2) in significantly less run-time (in hours or, in many cases, just minutes or seconds rather than several days), and (3) on a regular Desktop machine.

Acknowledgements

The authors would like to thank Igor Markov for many fruitful discussions. Furthermore, this works has partially been supported by the European Union through the COST Action IC1405 and the Google Research Award Program.

References

[1] The GNU MPFR library. http://www.mpfr.org.

[2] IBM Q. https://www.research.ibm.com/ibm-q/.

[3] S. Beauregard. Circuit for Shor's algorithm using 2n+3 qubits. *Quantum Information & Computation*, 3(2):175–185, 2003.

[4] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Computers*, 35(8):677–691, 1986.

[5] R. Courtland. Google aims for quantum computing supremacy. *IEEE Spectrum June 2017.*

[6] S. Debnath, N. Linke, C. Figgatt, K. Landsman, K. Wright, and C. Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536(7614):63–66, 2016.

[7] R. Drechsler, J. Shi, and G. Fey. Synthesis of fully testable circuits from BDDs. *IEEE Trans. on CAD*, 23(3):440–443, 2004.

[8] L. Gomes. Quantum computing: Both here and not here. *IEEE Spectrum April 2018.*

[9] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron. Quipper: a scalable quantum programming language. In *Conference on Programming Language Design and Implementation*, pages 333–342, 2013.

[10] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Symposium on the Theory of Computing*, pages 212–219, 1996.

[11] T. Häner, D. S. Steiger, M. Smelyanskiy, and M. Troyer. High performance emulation of quantum circuits. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, page 74, 2016.

[12] D. Hanneke, J. Home, J. Jost, J. Amini, D. Leibfried, and D. Wineland.

Realization of a programmable two-qubit quantum processor. *Nature Physics*, 6(1):13–16, 2010.

[13] H. Hiraishi and H. Imai. BDD operations for quantum graph states. In *Reversible Computation - 6th International Conference, RC 2014, Kyoto, Japan, July 10-11, 2014. Proceedings*, pages 216–229, 2014.

[14] N. Khammassi, I. Ashraf, X. Fu, C. Almudever, and K. Bertels. QX: A high-performance quantum computer simulation platform. In *Design, Automation and Test in Europe*, 2017.

[15] N. M. Linke, D. Maslov, M. Roetteler, S. Debnath, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe. Experimental comparison of two quantum computing architectures. *Proceedings of the National Academy of Sciences*, page 201618020, 2017.

[16] S. Malik, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *Int'l Conf. on CAD*, pages 6–9, 1988.

[17] T. Monz, D. Nigg, E. A. Martinez, M. F. Brandl, P. Schindler, R. Rines, S. X. Wang, I. L. Chuang, and R. Blatt. Realization of a scalable Shor algorithm. *Science*, 351(6277):1068–1070, 2016.

[18] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge Univ. Press, 2000.

[19] P. Niemann, R. Datta, and R. Wille. Logic synthesis for quantum state generation. In *Int'l Symp. on Multi-Valued Logic*, pages 247–252. IEEE, 2016.

[20] P. Niemann, R. Wille, and R. Drechsler. Efficient synthesis of quantum circuits implementing Clifford group operations. In *Asia and South Pacific Design Automation Conf.*, pages 483–488, 2014.

[21] P. Niemann, R. Wille, and R. Drechsler. Equivalence checking in multi-level quantum systems. In *Int'l Conf. of Reversible Computation*, pages 201–215, 2014.

[22] P. Niemann, R. Wille, and R. Drechsler. Improved synthesis of Clifford+T quantum functionality. *Design, Automation and Test in Europe*, 2018.

[23] P. Niemann, R. Wille, D. M. Miller, M. A. Thornton, and R. Drechsler. QMDDs: Efficient quantum function representation and manipulation. *IEEE Trans. on CAD*, 35(1):86–99, 2016.

[24] P. Niemann, A. Zulehner, R. Wille, and R. Drechsler. Efficient construction of qmdds for irreversible, reversible, and quantum functions. In *Int'l Conf. of Reversible Computation*, pages 214–231. Springer, 2017.

[25] E. Pednault, J. A. Gunnels, G. Nannicini, L. Horesh, T. Magerlein, E. Solomonik, and R. Wisnieff. Breaking the 49-qubit barrier in the simulation of quantum circuits. *arXiv preprint arXiv:1710.05867*, 2017.

[26] M. Reed, L. DiCarlo, S. Nigg, L. Sun, L. Frunzio, S. Girvin, and R. Schoelkopf. Realization of three-qubit quantum error correction with superconducting circuits. *Nature*, 482(7385):382–385, 2012.

[27] V. Samoladas. Improved BDD algorithms for the simulation of quantum circuits. In *European Symposium on Algorithms*, pages 720–731, 2008.

[28] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[29] M. Smelyanskiy, N. P. D. Sawaya, and A. Aspuru-Guzik. qHiPSTER: The quantum high performance software testing environment. *CoRR*, abs/1601.07195, 2016.

[30] D. S. Steiger, T. Häner, and M. Troyer. ProjectQ: an open source software framework for quantum computing. *arXiv preprint arXiv:1612.08091*, 2018.

[31] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Improving gate-level simulation of quantum circuits. *Quantum Information Processing*, 2(5):347–380, 2003.

[32] G. F. Viamontes, I. L. Markov, and J. P. Hayes. High-performance quidd-based simulation of quantum circuits. In *Proceedings of the conference on Design, automation and test in Europe-Volume 2*, page 21354. IEEE Computer Society, 2004.

[33] G. F. Viamontes, I. L. Markov, and J. P. Hayes. *Quantum Circuit Simulation*. Springer, 2009.

[34] D. Wecker and K. M. Svore. LIQUi|>: A software design architecture and domain-specific language for quantum computing. *CoRR*, abs/1402.4467, 2014.

[35] R. Wille, D. Große, D. M. Miller, and R. Drechsler. Equivalence checking of reversible circuits. In *Int'l Symp. on Multi-Valued Logic*, pages 324–330, 2009.

[36] A. Zulehner and R. Wille. One-pass design of reversible circuits: Combining embedding and synthesis for reversible logic. *IEEE Trans. on CAD*, 2017.

Alwin Zulehner Alwin Zulehner (S'17) received his BSc and MSc degree in computer science from the Johannes Kepler University Linz, Austria in 2012 and 2015, respectively. He is currently a Ph.D. student at the Institute for Integrated Circuits at the Johannes Kepler University Linz, Austria. His research interests include design automation for emerging technologies, currently focusing on reversible circuits and quantum circuits. In these areas, he has published several papers on international conferences and journals such as the IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems (TCAD), Asia and South Pacific Design Automation Conference (ASP-DAC), Design, Automation and Test in Europe (DATE) and International Conference on Reversible Computation.

Robert Wille Robert Wille (M'06–SM'09) received the Diploma and Dr.-Ing. degrees in computer science from the University of Bremen, Bremen, Germany, in 2006 and 2009, respectively. He was with the Group of Computer Architecture, University of Bremen, from 2006 to 2015, and has been with the German Research Center for Artificial Intelligence (DFKI), Bremen, since 2013. He was a Lecturer with the University of Applied Science of Bremen, Bremen, Germany, and a Visiting Professor with the University of Potsdam, Potsdam, Germany, and Technical University Dresden, Dresden, Germany. Since 2015, he is a Full Professor with Johannes Kepler University Linz, Linz, Austria. His current research interests include the design of circuits and systems for both conventional and emerging technologies. In these areas, he has published over 200 papers in journals and conferences. Dr. Wille has served in Editorial Boards and Program Committees of numerous journals/conferences, such as the IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems (TCAD), Asia and South Pacific Design Automation Conference (ASP-DAC), Design Automation Conference (DAC), Design, Automation and Test in Europe (DATE) and International Conference on Computer Aided Design (ICCAD).