Matrix-Vector vs. Matrix-Matrix Multiplication: Potential in DD-based Simulation of Quantum Computations

Alwin Zulehner Robert Wille
Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
alwin.zulehner@jku.at robert.wille@jku.at
http://iic.jku.at/eda/research/quantum/

***Abstract*—The simulation of quantum computations basically boils down to the multiplication of vectors (describing the respective quantum state) and matrices (describing the respective quantum operations). However, since those matrices/vectors are exponential in size, most of the existing solutions (relying on arrays for their representation) are either limited to rather small quantum systems or require substantial hardware resources. To overcome these shortcomings, solutions based on decision diagrams (DD-based simulation) have been proposed recently. They exploit redundancies in quantum states as well as matrices and, by this, allow for a compact representation and manipulation. This offers further (unexpected) potential. In fact, simulation has been conducted thus far by applying one operation (i.e. one matrix-vector multiplication) after another. Besides that, there is the possibility to combine several operations (requiring a matrix-matrix multiplication) before applying them to a vector. But since, from a theoretical perspective, matrix-vector multiplication is significantly cheaper than matrix-matrix multiplication, the potential of this direction was rather limited thus far. In this work, we show that this changes when decision diagrams are employed. In fact, their more compact representation frequently makes matrix-matrix multiplication more beneficial—leading to substantial improvements by exploiting the combination of operations. Experimental results confirm the proposed strategies for combining operations lead to speed-ups of several factors or—when additionally exploiting further knowledge about the considered instance—even of several orders of magnitudes.**

I. Introduction

Quantum computations utilize quantum mechanical effects like superposition and entanglement of so-called quantum bits (qubits) [1], that serve as basis for the so-called quantum parallelism. Computations that exploit these concepts to solve certain tasks significantly faster than conventional machines include the famous algorithms by Shor [2] (for integer factorization) and Grover [3] (for database search), as well as recently developed algorithms for quantum chemistry, machine learning, or solving large systems of linear equations [4].

Besides these theoretical algorithms, there has also been a huge process towards the physical realization of quantum computers. These achievements are not only driven by academia (cf. [5], [6], [7]), but also by large companies like IBM [8], Google [9], and Rigetti [10]—leading to a race for building the first quantum computer that is able to show quantum supremacy [11], [12]. Since all these realizations still suffer from the limited number of available qubits, a rather low fidelity (i.e. large error rates when applying operations), and small coherence time (i.e. how long a qubits state persists), researchers in this field still rely on simulators running on conventional machines.

Here, Schrödinger-style simulation is popular in which the respective computations are basically boiled down to the multiplication of vectors (describing quantum states) and matrices (describing quantum operations). Since both, vectors and matrices, are exponential in size with respect to the number of qubits, most of the existing solutions (relying on arrays for their representation) focus on tackling the underlying exponential complexity with massive hardware resources [13], [14], [15], [16], [17]. But even then, simulating arbitrary quantum computations with 46 qubits is today's limit [15].

To overcome these shortcomings, simulators based on *decision diagrams* (DDs) have been proposed recently [18], [19]. They aim for a more compact representation of the quantum states and operations by exploiting redundancies in the corresponding state vectors and operation matrices. While the worst case complexity still remains exponential, it has been shown for several practically relevant cases that they yield substantial performance improvements [19]—many instances that could not have been simulated before can be handled by DD-based simulation.

However, using DDs rather than e.g. arrays changes how the respective operations should actually be conducted. Thus far, quantum simulation has been conducted by multiplying a series of unitary matrices M_i (representing the quantum operations) to a vector v_0 (representing the initial state of the quantum system)—yielding a series of matrix-vector multiplications. Besides that, there is the possibility to combine several operations (requiring a matrix-matrix multiplication) before applying it to a vector. But since, from a theoretical perspective, matrix-vector multiplication is significantly cheaper than matrix-matrix multiplication, the potential of this direction was rather limited thus far (approaches such as proposed in [14], [20] used similar techniques but, after all, suffer from the limitations of array-based simulations as discussed above).

In this work, we re-visit this direction for DD-based simulation. To this end, we investigate how DDs perform during simulation. Our observations show that, despite the theoretical complexity, their more compact representation frequently makes matrix-matrix multiplication more beneficial. This offers (unexpected) potential which has not been exploited yet for DD-based simulation of quantum computations. Motivated by this, we are proposing several strategies that combine operations (using matrix-matrix multiplication) before a simulation step (using matrix-vector multiplication) is conducted. Experimental results confirm that this allows to accelerate DD-based simulation of quantum computations by several factors or, by additionally exploiting application-specific knowledge when combining operations, even several orders of magnitudes.

This paper is structured as follows. We recapitulate the main ideas of quantum computations and their DD-based simulation in Section II. In Section III, we analyze current DD-based simulation approaches and discuss their open potential that can be unveiled by combining operations. In Section IV, we propose strategies how to combine operations in order to exploit the available potential of DD-based simulation. Experimental evaluations summarized in Section V confirm that the proposed strategies indeed significantly improve the state of the art while Section VI concludes the paper.

II. Background

In this section, we review the basics of quantum computations and their DD-based simulation.

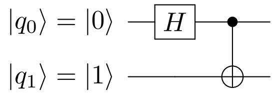


Fig. 1: Quantum circuit

A. Quantum Computations

In contrast to bits used in conventional computations, qubits cannot only be in one of the two basis states (denoted $|0\rangle$ and $|1\rangle$ using Dirac notation), but also in an (almost) arbitrary superposition of both, i.e. $|\psi\rangle = \alpha \cdot |0\rangle + \beta \cdot |1\rangle$. Measuring a qubit causes the superposition to collapse into one of the basis states $|0\rangle$ or $|1\rangle$ (with probabilities $|\alpha|^2$ and $|\beta|^2$). Extending this concept for n qubits, a quantum system results that is represented by 2^n complex amplitudes (one for each basis state) that form the so-called *state vector*. To satisfy the normalization constraint, their squared magnitudes have to sum up to 1.

The state of a quantum system can be modified by applying quantum operations, whose functionality are described by *unitary matrices*. Commonly used quantum operations acting on a single qubit are negation ($X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$), phase shift ($S = \begin{bmatrix} 1 & 0 \\ 0 & \mathrm{i} \end{bmatrix}$), or a so-called *Hadamard* operation which sets a qubit in superposition ($H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & \text{-}1 \end{bmatrix}$).

Quantum operations may be controlled by other qubits. Then, the operation is only conducted on the target qubit if the controlling qubits are in basis state $|1\rangle$. Otherwise, the identity is applied. One commonly used operation is the *controlled X* (CX) operation, whose functionality is represented by the unitary matrix

$$CX = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

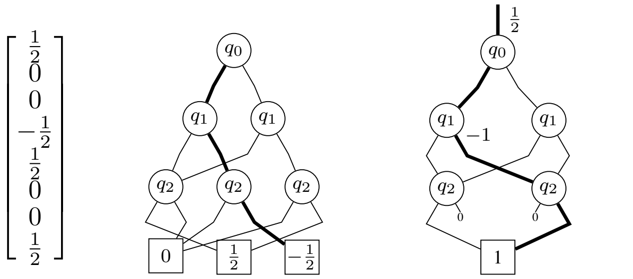
Quantum computations are often described by circuit diagrams, where qubits are represented by so-called *circuit lines* that are passed through a cascade of gates (representing the quantum operations). The gates indicate (from left to right) the order in which quantum operations are applied to the qubits.

Example 1. *The quantum circuit shown in Fig. 1 represents a quantum system composed of two qubits q_0 and q_1 with the initial state $|\psi\rangle = |01\rangle$. First, a Hadamard gate is applied to qubit q_0, then a CX gate with q_0 as controlling qubit (denoted by $\bullet$) and q_1 as target qubit (denoted by $\oplus$) is applied.*

Simulating quantum computations (i.e. determining the resulting state vector) requires to successively multiply the unitary matrices of the corresponding quantum gates to the state vector of the underlying quantum system.

Example 1 (continued)**.** *Since the first gate of the circuit acts only on qubit q_0, we assume the identity for qubit q_1. To adjust the size of the unitary matrix representing the quantum operation to the size of the state vector, one has to determine the Kronecker product $H \otimes I_2$. Since the size of the CX gate already matches the size of the state vector, the quantum circuit shown in Fig. 1 can be simulated by computing*

$$|\psi'\rangle = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{CX} \times \underbrace{\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & \text{-}1 & 0 \\ 0 & 1 & 0 & \text{-}1 \end{bmatrix}}_{H \otimes I_2} \times \underbrace{\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}}_{|01\rangle} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$



(a) Vector (b) DD w/o edge weights (c) DD with edge weights

Fig. 2: Representation for quantum states

B. DD-based Simulation of Quantum Computations

Since both, the size of the state vector as well as the size of the unitary matrices, grow exponentially with respect to the number of qubits,[1] researchers came up with approaches based on *decision diagrams* (DDs), which aim to gain a more compact representation by exploiting redundancies [18], [19]. In many cases, this leads to smaller memory consumption and faster simulation (cf. [18], [19]).

In DD-based simulation, state vectors are decomposed over their qubits. Consider the most significant qubit q_0 of a quantum state. All basis states with most significant qubit $q_0 = |0\rangle$ are located in the upper half of the state vector, whereas all basis states with $q_0 = |1\rangle$ are located in the lower half. Decomposing the vector into these two halves is represented by a DD node labeled q_0 with a left and a right successor (representing the upper and the lower half of the state vector, respectively). This decomposition is recursively applied until a single complex value is reached (represented by a terminal node holding the corresponding number). A compact representation is gained in many cases, since equal sub-vectors are represented by shared DD nodes.[2]

Example 2. *Fig. 2a shows the state vector of a quantum system composed of three qubits (q_0, q_1, and q_2). Fig. 2b shows the corresponding DD, which contains a terminal node (denoted by squared boxes) for each of the three different values in the state vector (0, $-\frac{1}{2}$, and $\frac{1}{2}$). The path from the root node to the terminal with value $-\frac{1}{2}$ highlighted in bold determines the amplitude for basis state $|q_0q_1q_2\rangle = |011\rangle$.*

To further increase sharing, the approach described in [19] utilizes weights that are attached to the edges of the DD.[3] This allows for extracting common factors, which are propagated upwards in the DD. By this, sub-vectors that are multiples of each other can be represented by the same DD node. The an entry of the state vector is then determined by the product of all edge weights on the corresponding path from the root node to the terminal.

Example 2 (continued)**.** *Fig. 2c shows the DD after adding weights to the edges. As can be seen, this leads to a more compact representation. For simpler graphical visualization of DDs, we denote zero vectors (i.e. vectors composed of 0-entries only) with 0-stubs in the DD and we omit edge weights that are equal to one. The path highlighted in bold again represents the amplitude of basis state $|q_0q_1q_2\rangle = |011\rangle$, which is now determined by computing $\frac{1}{2} \cdot 1 \cdot (-1) \cdot 1 = -\frac{1}{2}$.*

[1] Note that usually the matrices are not constructed explicitly since for many qubits the identity is assumed. Instead, smaller matrices applied exponentially many times to disjoint sub-vectors.

[2] Additionally, machine accuracy has to be taken into account as discussed in [21].

[3] A representation similar to *Quantum Multiple-valued Decision Diagrams* (QMDDs [22], [23]) results.

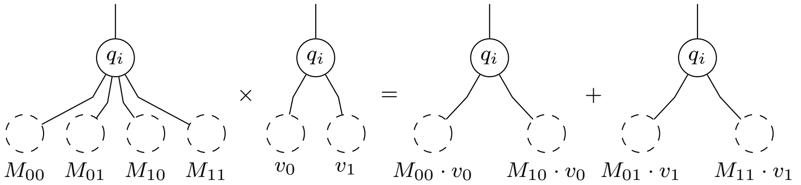


Fig. 3: Multiplication of a matrix and a state vectors

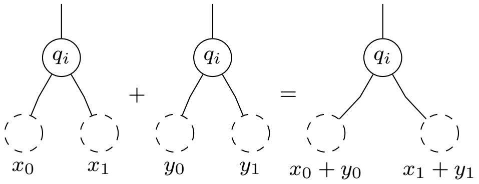


Fig. 4: Addition of state vectors

The idea described above can easily be extended—by adding a second dimension—to a description means for unitary matrices. This leads to four possibilities for the most significant qubit q_0—describing the four quadrants of the unitary matrix. Therefore, a DD node has four successor nodes which, from left to right, represent the left upper quadrant M_{00}, the right upper quadrant M_{01}, the left lower quadrant M_{10}, and the right lower quadrant M_{11}. Like for state vectors, edge weights allow to use shared nodes for representing sub-matrices that are multiples of each other—further reducing the overall number of DD nodes.

The multiplication of a vector and a matrix can then be conducted directly (and, thus, efficiently) on the DD representation by employing

$$M \times v = \begin{bmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{bmatrix} \times \begin{bmatrix} v_0 \\ v_1 \end{bmatrix} = \begin{bmatrix} M_{00} \times v_0 \\ M_{10} \times v_0 \end{bmatrix} + \begin{bmatrix} M_{01} \times v_1 \\ M_{11} \times v_1 \end{bmatrix}.$$

Fig. 3 sketches this: First, the four sub-products $M_{00} \cdot v_0$, $M_{01} \cdot v_1$, $M_{10} \cdot v_0$, and $M_{11} \cdot v_1$ are recursively determined (the respective sub-matrices and sub-vectors can easily be obtained by taking the pointers from the respective DD node). From these sub-products, two intermediate state vectors x and y are formed by adding DD nodes labeled with q_i (shown on the right hand side of Fig. 3). Finally, their sum is determined which eventually yields the final state vector. This addition is similarly conducted by employing

$$x + y = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} + \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} x_0 + y_0 \\ x_1 + y_1 \end{bmatrix}$$

as sketched in Fig. 4. Again, the resulting sub-sums $x_0 + y_0$ and $x_1 + y_1$ are determined recursively and combined by a new DD node.

III. Potential in DD-based Simulation

DD-based simulation approaches [18], [19] often allow to represent state vectors and matrices with much lower memory consumption (by exploiting redundancies as reviewed above) and, thus, allow to conduct simulation significantly faster. While state-of-the-art solutions introduced before (e.g. those proposed in [13], [14], [15], [16], [17]) required days and/or supercomputers for certain simulations, the DD-based simulators can handle those instances within minutes on a regular Desktop machine. However, in this section, we discuss in detail that there is still further potential which has not been exploited yet.

In general, the task of conducting a simulation of a quantum computation boils down to multiplying a series of unitary matrices M_i with $1 \leq i < g$ (representing a total of g quantum operations) to a vector v_0 (representing the initial state of the quantum system). That is, typically the resulting state vector v_g is determined by conducting a series of matrix-vector multiplications:

$$v_g = (M_g \times (M_{g-1} \times \cdots \times (M_1 \times v_0) \cdots)) \tag{1}$$

This simulation could also be done in a different fashion by rearranging parenthesis in Eq. 1 (since matrix-matrix multiplication is associative), but requires to also multiply matrices with each other. Then, a resulting state vector v_g is determined by conducting a series of matrix-matrix multiplications prior to a matrix-vector multiplication, i.e.

$$v_g = (M_g \times M_{g-1} \times \cdots \times M_1) \times v_0. \tag{2}$$

However, conventionally conducting a matrix-matrix multiplication turns out to be computationally more complex than conducting a matrix-vector multiplication, since even the best known algorithm specialized for multiplying square matrices (cf. [24]) has a complexity of $O(m^{2.373})$, while matrix-vector multiplication has a complexity $O(m^2)$.[4] Because of this, typically the scheme sketched in Eq. 1 is followed.

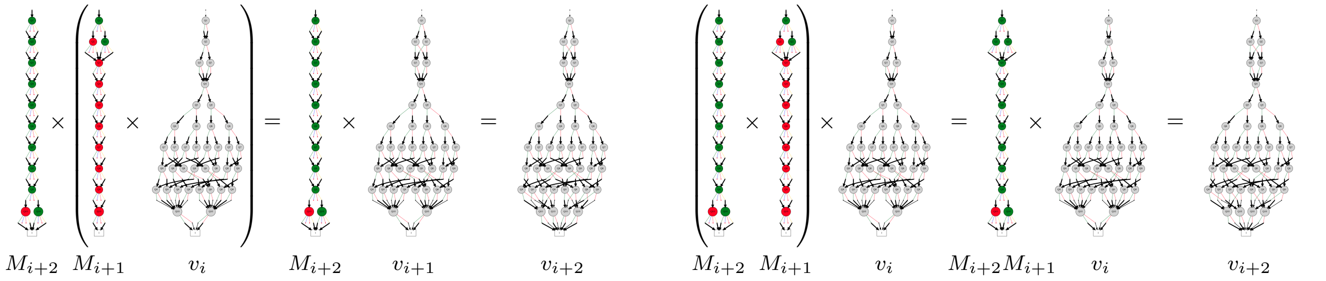
But this should change when simulations are conducted using DDs as they offer further potential. Although matrix-matrix multiplication in DDs also requires to recursively compute more sub-products and sub-sums per node compared to matrix-vector multiplication (as sketched before in Fig. 3 and Fig. 4), it has additionally to be taken into account that DDs representing elementary operations usually require significantly fewer nodes. This is because elementary operations work on one or two qubits only, while the remaining qubits just realize the identity I which can be represented by a single node for each qubit (i.e. in a linear fashion).[5] In contrast, vectors are usually rather complex after the first operations have been applied—leading to rather large DDs. This frequently leads to situations where, using DDs, a matrix-matrix multiplication is cheaper than a matrix-vector multiplication, although more steps per node are required (since re-occurring sub-products only have to be computed once). Thus, multiplying some small (with respect to the number of nodes) matrices first and multiplying the resulting product to a large vector only once (i.e. partially following Eq. 2) may be cheaper than sequentially multiplying a large vector to a sequence of small matrices (i.e. following Eq. 1).

Example 3. *Fig. 5 shows the DDs representing an intermediate state vector v_i as well as two elementary operations M_{i+1} and M_{i+2} (taken from a run simulating a circuit proposed by researchers from Google to demonstrate quantum supremacy [11]) as well as the intermediate results when conducting a corresponding simulation following Eq. 1 (depicted in Fig. 5a) and following Eq. 2 (depicted in Fig. 5b).*[6] *As can clearly been seen, following the established simulation flow with the supposedly cheaper matrix-vector multiplications, rather large DDs (i.e. the intermediate state vectors v_i and v_{i+1}) have to be processed in both of the*

[4] Note that, when considering quantum simulation as done here $m = 2^n$ (where n is the number of qubits).

[5] Details on how to efficiently construct such DDs are provided in [25].

[6] Note that, for the purpose of this example, it is not essential to provide all the details of the DDs (which is hard due to space constraints), but it is sufficient to get an intuition of the size of them.



(a) Conducting $v_{i+2} = M_{i+2} \times (M_{i+1} \times v_i)$ (i.e. following Eq. 1) (b) Conducting $v_{i+2} = (M_{i+2} \times M_{i+1}) \times v_i$ (i.e. following Eq. 1)

Fig. 5: Computational effect of rearranging parenthesis when computing $v_{i+2} = M_{i+2} \times M_{i+1} \times v_i$

two multiplications. In contrast, conducting the supposedly more expensive matrix-matrix multiplication first to combine the two rather small DDs requires to operate on a large DD (representing the vector) only once—significantly reducing the overall computational cost.

IV. Exploiting the Potential for More Efficient DD-based Simulation

Thus far, the fact that matrix-matrix multiplication might be cheaper than matrix-vector multiplication has not been exploited in DD-based simulation. At the same time, naively relying on matrix-matrix multiplication only (i.e. *completely* following Eq. 2) does not necessarily yield to an improvement (since, after all, the resulting representation of the intermediate matrices will grow as well). Accordingly, a compromise between the extreme cases (completely following Eq. 1 or Eq. 2) is required. This leaves the question how many and what operations shall be combined (by matrix-matrix multiplication) before another simulation step (i.e. matrix-vector multiplication) is applied.

In this section, we are presenting solutions to this question. To this end, we are proposing general strategies utilizing the potential motivated in Section III as well as strategies which additionally take further knowledge about the quantum computation to be simulated into account. Experimental evaluations summarized afterwards in Section V demonstrate the substantial impact of exploiting the discussed potential with these strategies.

A. General Strategies for Combining Operations

First, we propose general strategies that utilize the potential observed above. To decide how many operations shall be combined, the respective efficiency of DD-based simulation is taken into account. This efficiency usually depends on (1) the fact that the size of a DD representing a product of operations usually grows with the number of its factors (i.e. the number of combined operations) and (2) the fact that the costs of a multiplication heavily depends on the size of the DD representing the matrices and vectors. These observations motivate the following two strategies for combining operations:

- The first one (denoted *k-operations* in the following), forms the product of k operations before multiplying the resulting unitary matrix to the state vector. This directly takes the first observation into account. However, the size of the resulting product might be very small for one sequences of k operations, but huge for another sequence of k operations.
- The second strategy (denoted *max-size* in the following) avoids this problem and combines operations with respect to the size of the resulting DD. More precisely, operations are combined until their product exceed a certain size (defined by a parameter s_{max}). Then, the resulting matrix is multiplied to the state vector. Accordingly, parametrization is not with respect to just the pure number of operations but with respect to how large the corresponding DD get.

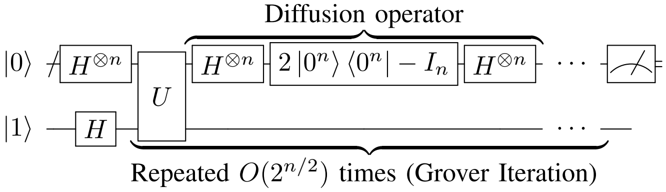


Fig. 6: Quantum circuit sketching Grover's Algorithm

Overall, both strategies aim for reducing the number of matrix-vector multiplications at the expense of increasing the number of matrix-matrix multiplications. For simulations, where the intermediate state vectors are composed of a large number of DD nodes, we expect a reduction of the time required to simulate quantum computations since these large DDs is not involved in all multiplications.

B. Strategies Utilizing Further Knowledge

The two strategies proposed above ignore the actual quantum computation to be simulated and completely rely on general parameters. Besides that, further potential can be realized if knowledge about the quantum computation to be simulated is taken into account.

For example, there exist several quantum algorithms where identical sub-circuits are repeated several times. Grover's algorithm for database search where a so-called Grover Iteration is conducted several times is a well-known example for this. Fig. 6 sketches the respective procedure in terms of quantum circuit notation. Here, a database U is concurrently queried with 2^n inputs (achieved by setting n qubits in superposition using an H-operation) and, afterwards, a diffusion operator is repeatedly applied to increase the probability of the desired database entry (the Grover Iteration). To maximize the probability of getting the desired database entry, one has to repeat the Grover iteration $2^{n/2}$ times—leading to a quadratic speed-up compared to a conventional algorithm.

When simulating such a quantum algorithm, the repeating sequence of operations obviously does not have to be considered completely from scratch each and every time. Instead, this constitutes a perfect sequence to be utilized for combining operations. More precisely, rather than consider elementary operations in each iteration, we combine all operations of a single iteration first (yielding a DD representing the entirety of a single iteration). Then, for each further iteration only the

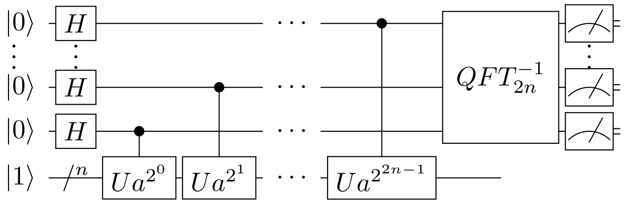


Fig. 7: Quantum circuit realizing Shor's algorithm

current state vector has to be multiplied with this combined matrix representation. This does not only save computation efforts because of the effects discussed above (by means of Example 3 and Fig. 5), but also because a sequence can—once it is pre-computed for the first iteration—be easily re-used for all further iterations—without the need of conducting any further matrix-matrix multiplications to combine operations. In the following, this strategy is denoted *DD-repeating*.

Moreover, even if a certain sequence of operations only occurs once, knowledge about the nature of the considered quantum computation still can help. In fact, many quantum algorithms include large Boolean parts (also denoted *oracles*) for which several different variations of sequences of (elementary) operations exist. Choosing and combining those operations in a fashion which suits DD-based simulation (and not in a fashion given by the quantum algorithm) can lead to further speed-ups.

Shor's algorithm [2] is a good example for that. This algorithm translates the problem of factorizing an n-bit number N to the problem of determining the multiplicative order r of another number a that is co-prime to N, i.e. r shall satisfy $a^r \equiv 1 \mod N$. To solve this problem on a quantum computer, one can use the circuit shown in Fig. 7, where modular exponentiation is performed conditionally on an n-qubit register (the Boolean components Ua^{2^i} compute $x \times 2^{a^i} \mod N$ for an input x). Eventually, an inverse *Quantum Fourier Transform* (QFT) is conducted in order to determine r with high probability.[7]

When executing this algorithm on a real quantum computer, the Boolean components are *decomposed* into elementary quantum operations. This yields rather complex sequences of elementary operations and even requires the addition of further so-called working (or ancillary) qubits. For example, the realization proposed in [26] requires $n+1$ such working qubits—resulting in a total number of $2n+2$ qubits for factoring an n-bit number. Instead, without breaking down the oracles, no working qubits and, hence, only $n+1$ qubits are required. Since it makes no difference for the quality of simulation whether the original functionality of Boolean components or the decomposed version is considered, combining their operations and additionally realizing them in a fashion which is suited for DD-based simulation is a promising strategy. In fact, constructing the DD for this functionality not through elementary operations but in a direct fashion allows to significantly reduce the number of matrix-matrix multiplications but also leads to a less number of qubits to be considered (yielding exponential improvements). In the following, this strategy is denoted *DD-construct*.[8]

V. Experimental Results

The strategies proposed above for exploiting further potential of DD-based simulation of quantum computation has been implemented on top of the state-of-the-art DD-based simulator [19]. Afterwards, we compared the respectively obtained results to those from the original implementation.[9] As benchmarks, we used established quantum computations, i.e. several instances of Shor's Algorithm [2] (using the implementation provided by Beauregard [27]) and Grover's Algorithm [3], as well as circuits proposed by researchers from Google to demonstrate quantum supremacy [11].[10] In this section, we summarize the findings of those evaluations.

First, we consider the results obtained by applying the general strategies as proposed in Section IV-A. Fig. 8 and Fig. 9 provide the speed-ups obtained when following strategy *k-operations* and strategy *max-size* compared to the original DD-based simulation, respectively. Here, the x-axis indicates the respective values chosen for the parameters k and s_{max} (c.f. Section IV-A), while the y-axis indicates the respectively obtained speed-up. Colors indicate the respective benchmark and the line shows the average speed-up obtained for each value of k/s_{max}.

These results confirm the discussions conducted in Section III: Doing simulation using only matrix-vector multiplications (i.e. following Eq. 1) does not fully utilize the possible potential. Instead, combining operations (using the supposedly more expensive matrix-matrix multiplications) to a certain extent (i.e. setting k or s_{max} to a value which allows for a combination of operations) improves the run-time significantly. The results, however, also confirm that combining *all* operations (i.e. completely following Eq. 2) still is not a suitable option. At some point, the benefits sketched in Example 3 and Fig. 5 disappear and the DD representing the resulting matrix gets too large. Overall, speed-ups of up to a factor of 3 (for *k-operations*) and 4.5 (for *max-size*) can be observed in the best cases.

Moreover, both strategies even allow to simulate benchmarks (namely some of the *shor*-benchmarks) which could not been simulated using the state-of-the-art DD-based simulation within a time-limit of 2 CPU hours. Since those results cannot be presented in Fig. 8/Fig. 9 (the speed-up cannot be determined in case of a time-out), they are separately listed in the first three columns of Table II (the first column provides the name of the benchmarks, the second one the "run-time" of the state-of-the-art approach, and the third one the results obtained by the best choice of k/s_{max}). As can be seen, just by utilizing the general strategies presented in Section IV-A, the run-time for these benchmarks can be reduced from more than 2 hours to a couple of minutes.

In a second series of experiments, we evaluated the improvements gained by utilizing further knowledge as proposed in Section IV-B. As discussed there, this is applicable for the *grover*-benchmarks (in case of strategy *DD-repeating*) and the *shor*-benchmarks (in case of *DD-construct*). Table I and Table II list a representative subset of the correspondingly obtained results (in addition to the first three columns already described above, the forth column additionally gives the required run-time when the respective strategy is applied).

[7] Note that the inverse QFT can also be conducted on a single qubit by using intermediate measurements [26], [27].

[8] Note that a similar scheme called *emulation* has been conducted in [20] for simulation approaches not relying on DDs.

[9] For a comparison of DD-based simulation in general to other simulation approaches (e.g. [13], [14], [15], [16], [17]), we refer to [19].

[10] In the following, those benchmarks are denoted by *shor*, *grover*, and *supremacy*, respectively, followed by a number indicating the number of qubits in the computation. For Shor's algorithm, the name also includes the number N to be factored as well as the number a co-prime to N, since these numbers significantly affect the simulation time. That is, those benchmarks have the form *shor_N_a_qubits*. Similarly, Google's benchmarks have the form *supremacy_depth_qubits*.

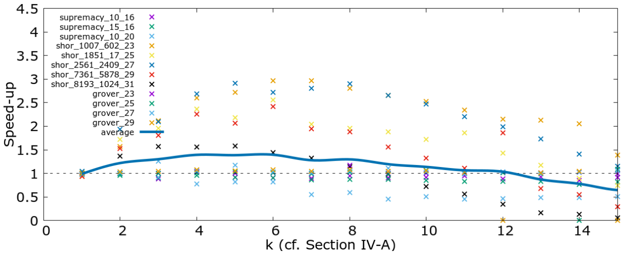


Fig. 8: Speed-up for strategy *k-operations*

TABLE I: Results for *grover*-benchm. (strategy *DD-repeating*)

Benchmark	t_{sota}	$t_{general}$	$t_{DD-repeating}$
Grover_23	13.77	4.83	2.78
Grover_25	31.63	11.77	6.23
Grover_27	72.95	26.84	14.25
Grover_29	169.05	67.82	30.87

TABLE II: Results for *shor*-benchm. (strategy *DD-construct*)

Benchmark	t_{sota}	$t_{general}$	$t_{DD-construct}$
shor_1007_602_23	84.74	19.72	0.12
shor_1851_17_25	94.99	31.08	0.13
shor_2561_2409_27	317.098	74.53	0.23
shor_7361_5878_29	159.48	49.41	0.14
shor_5513_3591_29	>7 200.00	217.20	0.66
shor_8193_1024_31	53.53	20.24	0.04
shor_11623_7531_31	>7 200.00	1423.56	3.05

As can be seen, the strategies discussed in Section IV-B yield substantial improvements. In case of the *grover*-benchmarks (additionally re-using a DD representing a combined sequence of operations after the first iteration), further speed-ups up to a factor of 2 can be achieved. In case of the *shor*-benchmarks (additionally utilizing a more suited DD construction and particularly less qubits to represent), further speed-ups of several orders of magnitudes can be achieved. More precisely, this strategy frequently allows to boil down the run-time from over 2 CPU hours to just very few seconds or even less.

Overall, for a DD-based simulation approach (which already has shown its advantages compared to other established simulators such as [13], [14], [15], [16], [17]) substantial further potential could be identified and utilized.

VI. Conclusion

In this work, we unveiled further potential of DD-based simulation of quantum computations. More precisely, we utilized the fact that matrix-matrix multiplication is not necessarily more expensive than matrix-vector multiplication when using DDs for their representation (which is contrary to simulation approaches using array-based representations). Exploiting this observation allows to develop strategies for combining operations before applying them to the state vector—leading to speed-ups of several factors or, when additionally exploiting further knowledge, even of several orders of magnitude compared to the state of the art. An implementation of the proposed strategies is publicly available at http://iic.jku.at/eda/research/quantum_simulation.

Acknowledgements

This work has partially been supported by the European Union through the COST Action IC1405 and the Google Research Award Program.

References

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

[2] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Jour. of Comp.*, vol. 26, no. 5, pp. 1484–1509, 1997.

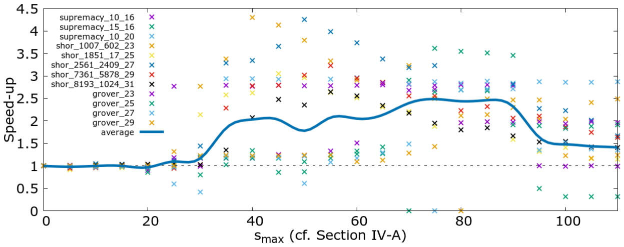


Fig. 9: Speed-up for strategy *max-size*

[3] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Symp. on Theory of Computing*, 1996, pp. 212–219.

[4] A. Montanaro, "Quantum algorithms: an overview," *npj Quantum Information*, vol. 2, p. 15023, 2016.

[5] S. Debnath, N. Linke, C. Figgatt, K. Landsman, K. Wright, and C. Monroe, "Demonstration of a small programmable quantum computer with atomic qubits," *Nature*, vol. 536, no. 7614, pp. 63–66, 2016.

[6] E. A. Martinez, C. A. Muschik, P. Schindler, D. Nigg, A. Erhard, M. Heyl, P. Hauke, M. Dalmonte, T. Monz, P. Zoller *et al.*, "Real-time dynamics of lattice gauge theories with a few-qubit quantum computer," *Nature*, vol. 534, no. 7608, pp. 516–519, 2016.

[7] 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*, p. 201618020, 2017.

[8] "IBM Q," https://www.research.ibm.com/ibm-q/, accessed: 2018-11-26.

[9] C. Neill, P. Roushan, K. Kechedzhi, S. Boixo, S. Isakov, V. Smelyanskiy, A. Megrant, B. Chiaro, A. Dunsworth, K. Arya *et al.*, "A blueprint for demonstrating quantum supremacy with superconducting qubits," *Science*, vol. 360, no. 6385, pp. 195–199, 2018.

[10] E. A. Sete, W. J. Zeng, and C. T. Rigetti, "A functional architecture for scalable quantum computing," in *Int'l Conf. on Rebooting Computing (ICRC)*, 2016, pp. 1–6.

[11] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, "Characterizing quantum supremacy in near-term devices," *Nature Physics*, vol. 14, no. 6, p. 595, 2018.

[12] R. Courtland, "Google aims for quantum computing supremacy," *IEEE Spectrum*, vol. 54, no. 6, pp. 9–10, 2017.

[13] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron, "Quipper: a scalable quantum programming language," in *Conf. on Programming Language Design and Implementation*, 2013, pp. 333–342.

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

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

[16] 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.

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

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

[19] A. Zulehner and R. Wille, "Advanced simulation of quantum computations," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2018.

[20] T. Häner, D. S. Steiger, M. Smelyanskiy, and M. Troyer, "High performance emulation of quantum circuits," in *Int'l Conf. for High Performance Computing, Networking, Storage and Analysis*, 2016, p. 74.

[21] A. Zulehner, P. Niemann, R. Drechsler, and R. Wille, "Accuracy and compactness in decision diagrams for quantum computation," in *Design, Automation and Test in Europe*, 2019.

[22] P. Niemann, R. Wille, and R. Drechsler, "On the "Q" in QMDDs: Efficient representation of quantum functionality in the QMDD data-structure," in *Int'l Conf. of Reversible Computation*, 2013, pp. 125–140.

[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 of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 86–99, 2016.

[24] F. Le Gall, "Powers of tensors and fast matrix multiplication," in *Int'l. Symp. on Symbolic and Algebraic Computation*. ACM, 2014, pp. 296–303.

[25] 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*. Springer, 2017, pp. 214–231.

[26] T. Häner, M. Roetteler, and K. M. Svore, "Factoring using $2n+2$ qubits with toffoli based modular multiplication," *Quantum Information & Computation*, vol. 17, no. 7&8, pp. 673–684, 2017.

[27] S. Beauregard, "Circuit for Shor's algorithm using 2n+3 qubits," *Quantum Information & Computation*, vol. 3, no. 2, pp. 175–185, 2003.