Concurrency in DD-based Quantum Circuit Simulation

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

Abstract—Despite recent progress in physical implementations of quantum computers, a significant amount of research still depends on simulating quantum computations on classical computers. Here, most state-of-the-art simulators rely on array-based approaches which are perfectly suited for acceleration through concurrency using multi- or many-core processors. However, those methods have exponential memory complexities and, hence, become infeasible if the considered quantum circuits are too large. To address this drawback, complementary approaches based on decision diagrams (called *DD-based simulation*) have been proposed which provide more compact representations in many cases. While this allows to simulate quantum circuits that could not be simulated before, it is unclear whether DD-based simulation also allows for similar acceleration through concurrency as array-based approaches. In this work, we investigate this issue. The resulting findings provide a better understanding about when DD-based simulation can be accelerated through concurrent executions of sub-tasks and when not.

I. Introduction

Quantum computers [1] provide a promising technology which bears a disruptive potential for many areas. For example, quantum algorithms to factorize integers (Shor's algorithm [2]) or to conduct database search (Grover's algorithm [3]) are capable of breaking limits imposed by today's conventional technologies—even if practically relevant realizations are not within reach yet. Other applications such as quantum chemistry [4] may utilize their potential in the near future. In fact, quantum computers are currently used to find new, and more efficient catalysts for chemical processes such as ammonia production in the Haber-Bosch process—optimizing an application that consumes 2% of the worlds yearly energy supply. Besides that, machine learning, cryptography, quantum simulation or solving systems of linear equations are areas where quantum computers may prove very beneficial as well [5]–[7].

The momentum of quantum computers is further shown by impressive accomplishments in their physical implementations and ongoing commercialization. Recently, IBM presented the first commercially available quantum computer with 20 qubits [8]. Before, they already launched their project IBM Q—providing public access to quantum processors through cloud access [9]. Similarly, Google is working on a 72 qubit chip that it hopes will be able to demonstrate quantum supremacy [10]. And also further companies such as Intel, Rigetti, Microsoft, and Alibaba work on this technology.

However, quantum computing is still an emerging technology. Since physically available quantum computers are limited and restricted in their applications, a significant amount of research still depends on simulating quantum computations on classical computers. Furthermore, simulations based on classical technology are still necessary to verify current quantum computer implementations. And, finally, simulation capabilities provide an estimate on *quantum supremacy* [11]. Accordingly, how to efficiently simulate quantum circuits is an important research topic.

From a mathematical perspective, quantum circuit simulation boils down to several multiplications of vectors (representing the current state of the considered quantum system) and matrices (representing the respectively applied quantum operations). Accordingly, most state-of-the-art simulators use representations like 1- and 2-dimensional arrays for vectors and matrices, respectively (see, e.g., [12]–[17]). Since matrix-vector multiplication can be easily divided into independent sub-tasks, the performance of those methods can be significantly boosted by concurrency using multi- or many-core processors. Hence, the state-of-the-art in quantum circuit simulation heavily relies on concurrent computations.

At the same time, however, these methods suffer from memory explosion since both, the respective vectors and the matrices, grow exponentially with the number of considered qubits—severely limiting their application. In order to overcome this problem, complementary approaches have been proposed that aim for a more compact representation of those vectors/matrices by exploiting redundancies (which frequently occur). This led to quantum circuit simulators based on *Decision Diagrams* (DDs) [18]–[22] that allow to keep the memory requirement far below the theoretical maximum in many practically relevant cases. This enabled the simulation of several quantum computations that could not be simulated before and established DD-based quantum circuit simulation as a complementary alternative to array-based simulators.

However, while array-based simulations are perfectly suited for concurrency (and, as reviewed above, make substantial use of it), it is unclear whether DD-based simulation allows for similar improvements. In fact, the DD-based approaches introduced thus far only utilize a single CPU core and no investigation on concurrent DD-based quantum circuit simulation has been conducted yet—raising the question whether there is further potential in improving DD-based simulation through concurrency.

In this work, we shed light on this issue and investigate the potential and challenges of concurrent simulation of quantum circuits based on decision diagrams. To this end, we first recapitulate the basics of quantum computations as well as the mode of operation of state-of-the-art array-based simulators in Section II and Section III, respectively. Afterwards, Section IV reviews the concepts of DD-based simulation and discusses challenges that emerge when employing those concepts in a concurrent fashion—including the description of a corresponding concurrent implementation. The main goal of these investigations and discussions is to eventually gain an understanding about when DD-based simulation can be accelerated through concurrency and when not. The findings made are confirmed by evaluations on representative cases (including the simulation of Shor's algorithm, Grover's algorithm, a quantum chemistry instance, and quantum supremacy instances) which are discussed in Section V. Finally, the paper is concluded in Section VI.

II. Quantum Computing

Computations in the quantum realm utilize *quantum bits* (*qubits*) [1], which can assume more states than just 0 and 1 known from conventional logic. In fact, while the basis states (denoted by $|0\rangle$ and $|1\rangle$ in Dirac-notation) are equal, a qubit $|\psi\rangle$ can be in any linear combination $|\psi\rangle = \alpha \cdot |0\rangle + \beta \cdot |1\rangle$ described through *amplitudes* $\alpha, \beta \in \mathbb{C}$ with $|\alpha|^2 + |\beta|^2 = 1$. If both amplitudes α and β are non-zero, the qubits state is also referred to as being in *superposition*. A second exploitable quantum effect is called *entanglement*, where the measurement of a qubit influences the state of another qubit. On a quantum computer, the values of α and β cannot be observed. Instead, a measurement of a qubit collapses the state of the qubit back to one of the basis states $|0\rangle$ (with probability $|\alpha|^2$) and $|1\rangle$ (with probability $|\beta|^2$). Afterwards, superposition of the measured qubit is destroyed. For systems with more than one qubit, the amplitudes apply to each of the possible basis states, e.g., for two qubits $|\psi\rangle = \alpha_0 \cdot |00\rangle + \alpha_1 \cdot |01\rangle + \alpha_2 \cdot |10\rangle + \alpha_3 \cdot |11\rangle$, where the sum $\sum_i |\alpha_i|^2$ has to equal 1. To allow a more compact representation, the amplitudes are commonly written as vector for all possible states, e.g., $\psi = [\alpha_0, \alpha_1, \alpha_2, \alpha_3]^\mathrm{T}$ for two qubits.

Example 1. *Consider a quantum system composed of two qubits which is in the state* $|\psi\rangle = 1/\sqrt{2}\,|00\rangle + 0\,|01\rangle + 0\,|10\rangle + 1/\sqrt{2}\,|11\rangle$. *This represents a valid state, since* $(1/\sqrt{2})^2 + 0^2 + 0^2 + (1/\sqrt{2})^2 = 1$. *The corresponding state vector is* $\psi = [1/\sqrt{2}, 0, 0, 1/\sqrt{2}]^\mathrm{T}$. *Measuring this system yields one of the two basis states* $|00\rangle$ *or* $|11\rangle$, *both with probability of* $|1/\sqrt{2}|^2 = 1/2$. *That is, after the measurement, the state vector is either* $\psi = [1, 0, 0, 0]^\mathrm{T}$ *or* $\psi = [0, 0, 0, 1]^\mathrm{T}$ *(each measured with a probability of* $1/2$*).*

The current state of a quantum system can be manipulated using quantum operations, which are inherently reversible with the exception of measurement. Quantum operations are defined through unitary matrices, i.e., square matrices whose inverse is their conjugate transposed [1]. Examples of important quantum operations on a single qubit are

$$NOT = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, H = \tfrac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \text{ and } Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},$$

where NOT negates the state of the qubit, H sets the qubit into superposition, and Z shifts the phase of the qubit. Besides that, quantum operations that span more than one qubit exist as well. The most prominent example is the $CNOT$ operation, which negates a *target qubit*, iff the chosen *control qubit* is in the state $|1\rangle$. This is defined through the matrix

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

The effect of a quantum operation (represented by a matrix) to a quantum state (represented by a vector) can be described through matrix-vector multiplication as illustrated in the following example:

Example 2. *Consider a quantum system composed of two qubits which is currently in state* $|\psi\rangle = |10\rangle$. *Performing a* $CNOT$ *operation yields*

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{CNOT} \times \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}}_{\psi} = \begin{bmatrix} 0+0+0+0 \\ 0+0+0+0 \\ 0+0+0+0 \\ 0+0+1+0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \equiv |11\rangle.$$

III. Concurrent Array-based Simulation

Utilizing concurrent calculations is a key aspect of today's state-of-the-art quantum circuit simulation approaches. This is due to the structure of the task which is well-suited for acceleration through concurrency. This section briefly reviews this aspect and the corresponding related work—providing a motivation for the following considerations in this work.

In general, simulating a quantum computation can be carried out in a straight-forward fashion on a classical computer by performing matrix-vector multiplication as illustrated before in Example 2. Since each quantum state and each quantum operation can be represented by a matrix and a vector, respectively, many existing solutions for quantum circuit simulation directly realize these multiplications through arrays (see, e.g., [12]–[17]). However, since most quantum operations work on a small number of qubits, their matrices frequently have to be expanded to match the size of a considered multi-qubit system. This can be conducted by applying the Kronecker-product with an appropriately sized identity matrix as illustrated by the following example:

Example 3. *Consider the application of a Hadamard-operation on a single qubit in a two-qubit system. To describe that, the* 2×2 *single-qubit operation has to be defined for a two-qubit system, i.e., in terms of a* 4×4 *matrix. Here, the resulting matrix describes the application of the Hadamard-operation on the first qubit while the second qubit remains unchanged afterwards—realized by the Kronecker product of* H *and* I_2, *i.e.,*

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

If the respectively expanded matrix is available, simulation can be conducted as illustrated above. Moreover, the structure of the task, i.e., the matrix-vector multiplication, additionally allows for substantial speedups if a concurrent consideration is employed. In fact, the multiplication can be decomposed into the following sub-tasks:

$$\begin{bmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{bmatrix} \times \begin{bmatrix} V_0 \\ V_1 \end{bmatrix} = \begin{bmatrix} M_{00} \times V_0 + M_{01} \times V_1 \\ M_{10} \times V_0 + M_{11} \times V_1 \end{bmatrix} \qquad (1)$$

Here, $M_{00} \times V_0$, $M_{01} \times V_1$, $M_{10} \times V_0$, $M_{11} \times V_1$, as well as the additions can be computed independently with no or only negligible synchronization overhead. Afterwards, the respectively obtained results can easily be combined to the final result. This allows for the utilization of several CPU cores in modern computers as well as the utilization of "supercomputers" providing means for highly parallelized computations. Accordingly, state-of-the-art methods for quantum circuit simulation exploit this potential and heavily rely on concurrent computations [12]–[17]. Due to the rather moderate synchronization overhead[1], this frequently yields speedups that scale with the number of utilized cores.

IV. Concurrent DD-based Simulation

State-of-the-art approaches for quantum circuit simulation as reviewed above can easily be accelerated through concurrent executions. However, they suffer from memory explosion since both, the state vectors as well as the operation matrices grow

[1] In fact, hardly any synchronization is required and most overhead is only caused by the management of tasks and threads.

exponentially with respect to the number of qubits[2]. This severely limits the application of those approaches for many relevant cases. In fact, even though concurrency may be heavily utilized, simulation is bounded by whether the matrices/vectors still can be represented by the available memory.

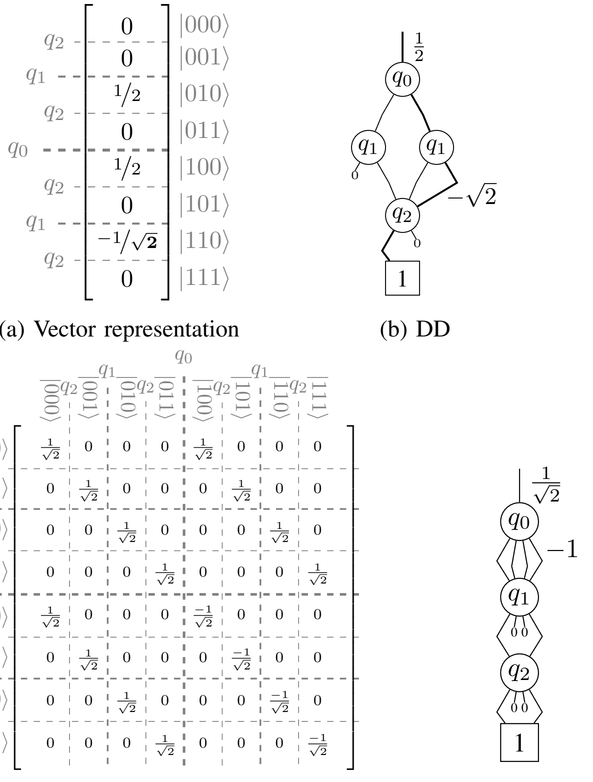
In order to address this problem, complementary approaches for quantum circuit simulation have been proposed which rely on decision diagrams (called *DD-based simulation* in the following; [18]–[22]). While this allows for a much more compact representation of matrices and vectors in many cases—allowing for simulations of quantum circuits that could not be simulated before—it also completely changes the way how simulation is conducted. This raises the question whether the benefits of concurrent computations as reviewed above can also be utilized for this complementary approach. In the following, we investigate this question. To this end, we first briefly review the basic concepts of DD-based simulation followed by a discussion of how this affects the possibilities to conduct those simulations in a concurrent fashion. Based on that, a concurrent implementation of DD-based quantum circuit simulation is presented.

A. DD-based Simulation

DD-based simulation has been proposed in [18]–[22] and rests on the main idea to represent matrices (vectors) in terms of decision diagrams [23] that allow to exploit redundancies in order to obtain a more compact representation. More precisely, consider a quantum system with qubits $q_0, q_1, \ldots, q_{n-1}$, whereby q_0 represents the most significant qubit. 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 two sub-vectors. By convention, the left (right) edge indicates the 0-successor (1-successor). The sub-vectors are recursively decomposed further until vectors of size 1 (i.e., complex numbers) result. During this decomposition, equivalent sub-vectors can be represented by the same nodes—reducing the complexity of the representation. Then, instead of having a terminal node for every distinct value in the state vector, common factors of the amplitudes are stored in the edge weights. The value can be reconstructed by multiplying the edge weights along the desired path in the decision diagram.

Example 4. *Consider the state vector in Fig. 1a—the annotations sketch how the vector is decomposed (left) and which base state corresponds to each entry in the vector (right). Fig. 1b shows the corresponding decision diagram. Here, e.g., the amplitude of the state $|110\rangle$ is accessed by following the path in the decision diagram for $q_0 = 1$, $q_1 = 1$, $q_2 = 0$ (bold lines) and multiplying the edge weights along the path, i.e., $1/2 \cdot 1 \cdot (-\sqrt{2}) \cdot 1 = -1/\sqrt{2}$.*

The representation of matrices follows a similar scheme. However, instead of splitting the matrix in two, it is split into four equal parts. Considering again a quantum system with qubits $q_0, q_1, \ldots q_{n-1}$ and a unitary matrix $U = [u_{i,j}]$. The matrix U is decomposed into four sub-matrices with dimension $2^{n-1} \times 2^{n-1}$: The entries in the sub-matrices provide the values describing the operation on the basis states in q_0 (top left $|0\rangle$ to $|0\rangle$; top right $|1\rangle$ to $|0\rangle$; bottom left $|0\rangle$ to $|1\rangle$; and bottom right $|1\rangle$ to $|1\rangle$). 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 this decomposition, equivalent sub-matrices are represented by the same nodes and a corresponding normalization scheme (as applied for the representation of state vectors) is employed. Note that for a simpler graphical notation, zero stubs are used to indicate zero matrices (i.e., matrices that contain zeros only) and edge weights equal to one are omitted.



$$\left[\begin{array}{c|c} 0 & |000\rangle \\ 0 & |001\rangle \\ 1/2 & |010\rangle \\ 0 & |011\rangle \\ 1/2 & |100\rangle \\ 0 & |101\rangle \\ -1/\sqrt{2} & |110\rangle \\ 0 & |111\rangle \end{array}\right]$$

(a) Vector representation

(b) DD

$$\begin{array}{c|cccccccc} & |000\rangle & |001\rangle & |010\rangle & |011\rangle & |100\rangle & |101\rangle & |110\rangle & |111\rangle \\ \hline |000\rangle & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ |001\rangle & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 \\ |010\rangle & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 \\ |011\rangle & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} \\ |100\rangle & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{-1}{\sqrt{2}} & 0 & 0 & 0 \\ |101\rangle & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{-1}{\sqrt{2}} & 0 & 0 \\ |110\rangle & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{-1}{\sqrt{2}} & 0 \\ |111\rangle & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{-1}{\sqrt{2}} \end{array}$$

(c) Matrix representation

(d) DD

Fig. 1: DD representations of state vector and matrix

Example 5. *Fig. 1c shows the matrix representing the quantum operation $U = H \otimes I_2 \otimes I_2$. The same operation is depicted in Fig. 1d as decision diagram.*

Having those representations for quantum states (vectors) and quantum operations (matrices), quantum circuit simulation can be conducted as illustrated on top of Fig. 2. This basically follows the scheme as provided in Eq. (1) above, i.e., the given matrix M and vector V are split into sub-matrices M_{00}, M_{01}, M_{10}, M_{11} and sub-vectors V_0, V_1. This is recursively conducted until terminal nodes in the decision diagrams are reached. During this process redundancies are exploited to avoid unnecessary computations, i.e., operations with identical operands are only performed once—the main difference to array-based simulations. The complexity is therefore not bounded by the number of entries in the matrix and the state vector, but by the number of nodes in their respective representations. While this still leads to an exponential complexity in the worst case, in many cases complexity and memory requirements can be significantly reduced. This allowed for significant speedups, e.g., from 30 days to 2 minutes or for the simulation of quantum circuits that could not been simulated before [20].

[2] Actual implementations (such as [12]) may realize the operation in a different way, but still require an exponential amount of memory to store the state vector.

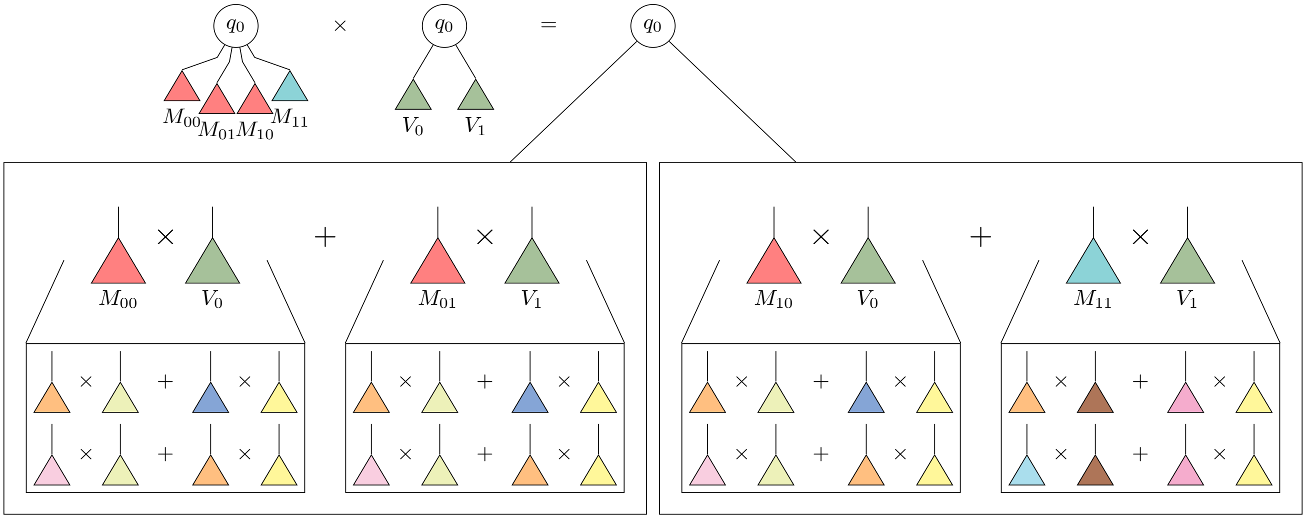


Fig. 2: Sketch of DD-based matrix-vector multiplication with operation M and state V

B. Concurrency in DD-based Simulation

Apparently, DD-based simulation offers significant advantages compared to array-based simulation. However, all implementations of DD-based simulators conduct the respective tasks in a sequential fashion. To date, it remains unknown whether accelerations through concurrency are possible in a similarly simple way as shown above with array-based simulators. In general, the main approach is the same: As shown before by means of Eq. (1) for array-based simulation, DD-based simulation decomposes the matrix-vector multiplication into sub-tasks. This is sketched in Fig. 2. However, as it turns out, the amount of sharing which can be exploited significantly affects the possible potential of a concurrent consideration of DD-based simulation.

More precisely, if no sharing is exploited at all (similar to array-based simulation), all sub-tasks can be independently considered—yielding perfect conditions for utilizing concurrency, but also requiring an exponential amount of memory. In contrast, if sharing is exploited (as sketched by colors in Fig. 2, i.e., sub-DDs which represent the same sub-matrix/sub-vector are colored equally), those conditions drastically deteriorate. While sharing allows for substantial reductions in the needed amount of memory (all sub-DDs with the same color need to be present in memory only once), they also destroy the independence of the sub-tasks. This affects the potential of concurrent DD-based simulation particularly in two ways:

- Substantially more processing time has to be spent on synchronization: When using the same memory region for two or more threads, proper synchronization is required to guarantee reading and writing the correct values in case of simultaneous access. That is, concurrent executions in DD-based simulations first have to deal with a substantial overhead to overcome before any improvements might be possible.
- Besides that, improvements are not always guaranteed. In fact, due to sharing, it may happen that identical calculations are simultaneously conducted on several cores, obviously leading to the same sub-DD. In these cases, there is no benefit in performance compared to a sequential DD-based approach, since those cases can also be covered by caching (in other words, a sequential calculation is able to identify previously conducted calculations, e.g., by means of *compute tables* [24] and, based on that, would never start the tasks for certain calculations which eventually would lead to the same sub-DDs again). For example, in the calculations sketched in Fig. 2, three threads might be busy recursively determining the results for $M_{00} \times V_0$, $M_{01} \times V_1$, and $M_{10} \times V_1$, while all results eventually turn out to be equivalent (a sequential approach would detect that after determining the result for the first sub-DD and would not start calculations for the other two, i.e., parallelization provides no improvement in this scenario).

Hence, a concurrent consideration of DD-based simulation would not only cause substantial overhead for managing the threads and keeping memory consistent, it additionally is not even guaranteed that this eventually does increase the performance—in particular in cases which provide lots of potential for sharing (usually wanted in order to reduce memory consumption but suddenly posing a problem for a concurrent consideration). Having this understanding, the following challenges need to be addressed when aiming for concurrent DD-based simulation:

How to efficiently access results from previous calculations?: In sequential simulation, results of previous calculations are effortlessly exploited by storing these results in the *compute table* and consulting this table before starting the next calculation. Concurrent execution on the other hand has to frequently synchronize the corresponding read and write operations to the compute table—leading to a potential bottleneck, which becomes worse as the number of utilized cores increases.

How to avoid computing identical operations more than once at the same time during concurrent execution?: While the previous challenge in essence also occurs in sequential DD-based simulation (although with much less overhead), avoiding the simultaneous computation of identical tasks is novel to the concurrent approach. If more than one task is executed simultaneously and the result of the sub-task is not yet stored in the compute table, several threads may run identical calculations—wasting processing time that could instead be utilized to compute other tasks. This voids the advantage of concurrent execution compared to a sequential approach.

How to determine the order in which tasks are executed?: As sketched in Fig. 2, the task graph can grow exponentially when sharing is not possible, thus completely storing all (sub-)tasks requires an exponential amount of memory with respect to the number of qubits. This can be mitigated by employing a scheduling which executes the tasks in an order that minimizes the number of tasks kept in memory at once. Ideally, only the currently executed tasks are kept in memory. However, this is not possible, as the depending tasks have to be kept as well. Moreover, the scheduling heavily influences the aforementioned challenges. If simultaneous execution of tasks on the same level can be avoided during scheduling, performing the same task on several cores will be avoided as well. Unsuitable scheduling strategies on the other hand will load every task into memory, i.e., resulting again in exponential memory usage.

In the following, strategies to tackle these challenges are presented. Based on them, a study on representative cases for quantum circuit simulation evaluates the possible benefit (or drawbacks) of a concurrent consideration of DD-based simulation.

C. Implementation

In this section, we present an implementation of a concurrent DD-based quantum circuit simulation. To this end, a sequential version (based on [25]) has been taken as starting point and equipped with the capability to execute the single tasks in a concurrent fashion. In order to address the challenges discussed above, the following solutions have additionally been employed:

Caching completed results: Caching the results of completed calculations in terms of a compute table provides a significant advantage in current (sequential) DD-based simulation. Therefore, it is desirable to retain this advantage as best as possible in a concurrent implementation. Since the compute table is a global structure, the reads and writes have to be synchronized—usually a severe bottleneck in concurrent executions. Furthermore, before each task is executed, the thread has to read from the compute table once and, if the result was not available, the task has to write it back after the calculation. In order to reduce the resulting synchronization efforts, we tried to “localize” the global structure of the compute table a bit. More precisely, we used several independent instances of the compute table: one for each type of operation. This allows to simultaneously write the results of tasks for different operations, e.g., multiplication and addition, into different compute tables and, by this, reduces the synchronization overhead.

Waiting for ongoing calculations: Unnecessarily performing identical calculations on several cores as discussed above should be avoided. While the solution to this challenge is conceptionally straight-forward (simply check if the next task is currently executed by another thread), the actual implementation is more complicated as this requires extended communication between the involved threads. Our implementation uses a synchronized data structure that maps from a tuple of operation and the respective operands to a flag indicating if this task is currently executed. Before a thread creates a sub-task, it does not only check if the result is already available, but also if the sub-task is marked as currently running as a dependency of another task. If this is the case, the thread is suspended until the result becomes available.

Naturally, this check comes with a runtime overhead for synchronization due to the extended communication required to suspend a thread and, then, resume its work at an appropriate time. Compared to the compute table, this scheme requires at least one read operation (more if the thread is suspended) and two write operations for each task (flagging the task as currently running and finally removing this flag). Measurements showed that this additional synchronization imposes a significant overhead—leading to worse runtime behavior, even more so in combination with the scheduling described next.

To overcome this problem, the concurrent execution of tasks operating on the same variable should be avoided. Moreover, tasks which are close to the terminal nodes are computationally less expensive than the locking mechanism, i.e., in these cases the check and suspend procedure should be dropped (even if it leads to concurrent calculations of the same tasks). Overall, this is realized by scheduling the execution of tasks in a depth first fashion up to a certain level.

Scheduling using priorities: As discussed above, a clever scheduling is capable of mitigating several of the aforementioned problems. To achieve a strategy akin to depth first traversal, the tasks are assigned according to a priority which corresponds to the depth in the task graph, i.e., the sub-tasks have a higher priority than their parent task. This avoids a fan out where tasks on the same level are executed simultaneously (akin to breadth-first traversal), which would result in an exponential amount of tasks in memory at the same time.

In our implementation, the priority of a task is not determined by traversing the task graph, but rather by tracking the indices i of the nodes q_i. Therefore operations involving nodes farther from the root node, i.e., $i = 0$, are preferred when scheduling the next task to be executed. Additionally, for nodes of the same level (i.e., the same i), multiplication is executed preferably over addition. This scheme eventually employs a depth-first-like traversal, where tasks farther from the root node are executed first.

These solutions eventually aim to address and/or mitigate the challenges of DD-based quantum circuit simulation discussed above. However, they cannot completely avoid the principle obstacles that are caused by DDs using sharing and, hence, require substantial synchronization while, at the same time, may provide limited potential for improvements due to the fact that tasks are executed which yield the same result. Hence, how concurrent DD-based simulation eventually performs is considered using representative examples in the next section.

V. Representative Cases

In order to investigate concurrent DD-based quantum circuit simulation also on some representative quantum algorithms, we implemented a concurrent simulator equipped with the strategies described in Section IV-C (based on [25]). To this end, we used C++ as programming language and OpenMP to introduce parallel execution. The simulations themselves were then conducted on a server providing a 64-bit machine with 32 cores running at 2.3 GHz and 64 GiB memory using Linux 4.13 as kernel.

As representatives, we used quantum algorithms from typical application areas currently considered for quantum computers, namely

- quantum algorithms factorizing integers (i.e., a realization of Shor’s factorization algorithm [2]; denoted by *shor_x*, where *x* represents the factored number),
- quantum algorithms conducting a database search (i.e., a realization of Grover’s search algorithm [26]; denoted by *grover_x*, where *x* represents the number of qubits in the oracle),
- a quantum algorithm implementing functionality from quantum chemistry (as provided in [27]; denoted by *qua_chem*), and
- quantum circuits used to conduct quantum supremacy experiments (as provided by researchers from Google [11]; denoted by *supremacy_5_4_x*, where x represents the depth of the circuit on a 5×4 surface).

The results are summarized in Fig. 3. Here, the graphs show the effect on the runtime for each algorithm when conducting a concurrent DD-based simulation (with up to 32 cores) compared to conducting a sequential DD-based simulation.

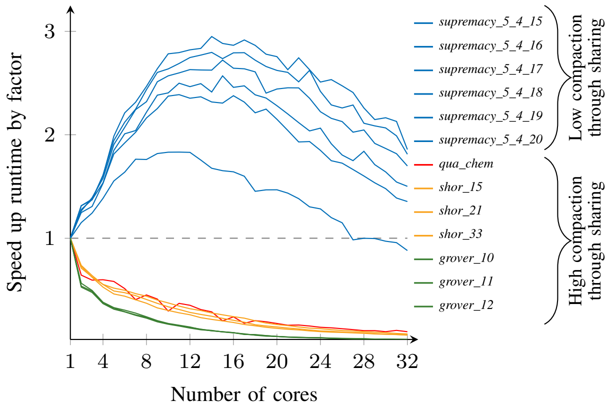


Fig. 3: Performance of concurrent DD-based simulation

Note that each group of algorithms is denoted with the same color. Furthermore, we categorized the respective instances according to the compaction which became possible through sharing (cf. Section IV). More precisely, almost no sharing and, hence, only low compaction is achievable for the supremacy algorithms. In contrast, high compaction through sharing is possible for the *shor*-, *grover*-, and *qua_chem*-instances. Those are also the instances where DD-based simulation offers substantial improvements compared to array-based simulation (exactly because of the compact representation).

The results clearly confirm the discussions from Section IV. If no or almost no sharing is possible in the DDs, sub-tasks can be independently considered—yielding better conditions for concurrent execution. In contrast, if a high degree of sharing is possible (yielding a much more compact representation and, hence, efficient simulation), no further gains can be realized through concurrent execution. Moreover, as already discussed above, this does not only lead to *no* further improvements, but the additional overhead for managing the threads and synchronizing memory accesses causes an actual decrease in the performance. In fact, in all of these cases no improvement but a significant degradation of performance is reported.

Overall, this confirms the understanding about concurrency of DD-based quantum circuit simulation as developed above: DD-based simulation offers great potential as an alternative to array-based simulation when lots of sharing is possible (then, DD-based simulation can overcome a possible memory explosion as already observed in [18]–[22]). In these cases, however, further accelerations due to concurrent executions become much harder. Vice versa, if no or only few sharing is possible, DD-based simulation can be accelerated through concurrent execution (basically, using the same principles as concurrent array-based simulation). In the cases considered here, improvements up to a factor of 3 (for the supremacy circuits using 14 cores) can be achieved.

VI. Conclusion

Quantum circuit simulation is an important research area where most of the existing solutions rely on array-based approaches. They can be perfectly accelerated by concurrent execution, but suffer from memory explosion problems. As an alternative, complementary solutions based on decision diagrams have been proposed. However, whether these approaches can also be nicely improved by a concurrent consideration of sub-tasks has not been investigated yet. In this work we shed light on this issue. We discussed the concepts of DD-based quantum circuit simulation and the corresponding challenges when aiming for a concurrent execution of the respective sub-tasks. This unveiled that the potential of concurrent DD-based simulation heavily depends on the amount of sharing exploited by the DDs. Evaluations on representative quantum algorithms confirmed these findings. By this, a better understanding about DD-based quantum circuit simulation and particularly its potential for acceleration through concurrency has been developed.

Acknowledgments

This work has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

References

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

[2] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," *Foundations of Computer Science*, pp. 124–134, 1994.

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

[4] Y. Cao, J. Romero, J. P. Olson, M. Degroote, P. D. Johnson, M. Kieferová, I. D. Kivlichan, T. Menke, B. Peropadre, N. P. Sawaya *et al.*, "Quantum chemistry in the age of quantum computing," *arXiv:1812.09976*, 2018.

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

[6] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[7] P. J. Coles, S. Eidenbenz, S. Pakin, A. Adedoyin, J. Ambrosiano, P. Anisimov, W. Casper, G. Chennupati, C. Coffrin, H. Djidjev *et al.*, "Quantum algorithm implementations for beginners," *arXiv:1804.03719*, 2018.

[8] "IBM unveils world's first integrated quantum computing system for commercial use," https://newsroom.ibm.com/2019-01-08-IBM-Unveils-Worlds-First-Integrated-Quantum-Computing-System-for-Commercial-Use, Accessed: 2019-04-05.

[9] "IBM Q," https://research.ibm.com/ibm-q/, Accessed: 2019-04-05.

[10] J. Kelly, "Engineering superconducting qubit arrays for quantum supremacy," in *APS Meeting Abstracts*, 2018, p. A33.001.

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

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

[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] A. Cross, "The IBM Q experience and QISKit open-source quantum computing software," in *APS Meeting Abstracts*, Mar 2018, p. L58.003.

[16] D. S. Steiger, T. Häner, and M. Troyer, "ProjectQ: an open source software framework for quantum computing," *Quantum*, vol. 2, p. 49, 2018.

[17] "Cirq: A python framework for creating, editing, and invoking Noisy Intermediate Scale Quantum (NISQ) circuits," https://github.com/quantumlib/cirq, Accessed: 2019-04-05.

[18] V. Samoladas, "Improved BDD algorithms for the simulation of quantum circuits," in *European Symposium on Algorithms*, 2008, pp. 720–731.

[19] G. F. Viamontes, I. L. Markov, and J. P. Hayes, "High-performance QuIDD-based simulation of quantum circuits," in *Design, Automation and Test in Europe*, 2004, p. 21354.

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

[21] A. Zulehner and R. Wille, "Matrix-vector vs. matrix-matrix multiplication: Potential in DD-based simulation of quantum computations," in *Design, Automation and Test in Europe*, 2019.

[22] 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.

[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] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," in *Design Automation Conf.*, 1990, pp. 40–45.

[25] A. Zulehner, S. Hillmich, and R. Wille, "How to efficiently handle complex values? implementing decision diagrams for quantum computing," *Int'l Conf. on CAD*, 2019, The implementation is available at http://iic.jku.at/eda/research/quantum_dd/.

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

[27] R. Babbush, N. Wiebe, J. McClean, J. McClain, H. Neven, and G. K.-L. Chan, "Low-depth quantum simulation of materials," *Phys. Rev. X*, vol. 8, p. 011044, 2018.