SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Demand-driven Entangling Weighted Decision Diagrams for Quantum Simulation

Andriy Manucharyan

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Demand-driven Entangling Weighted Decision Diagrams for Quantum Simulation

Bedarfsgesteuerte Verschränkung gewichteter Entscheidungsdiagramme für Quantensimulationen

Author:	Andriy Manucharyan
Supervisor:	Prof. Dr. Helmut Seidl
Advisor:	M.Sc. Yannick Stade
Submission Date:	15.08.2023

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.08.2023 Andriy Manucharyan

Acknowledgments

I would like to express my sincere gratitude and appreciation to my advisor, M.Sc. Yannick Stade without whom this work would not have been possible. His constant support and valuable insights helped me a lot during this research. I am particularly grateful for his consistent provision of constructive feedback and invaluable advice, which have been instrumental in refining the quality of this study. Also, I would like to thank Dr. Stefan Hillmich. His willingness to address relevant inquiries and share insights significantly contributed to the clarity and depth of this work. Furthermore, I extend my sincere gratitude to the comprehensive faculty and dedicated staff at the Technical University of Munich (TUM), with a particular acknowledgment to the Chair for Programming Languages, Compiler Construction and Specification Formalisms. Their generous provision of the opportunity to undertake this thesis is greatly appreciated.

Abstract

Quantum simulators have proven to be a useful tool in the field of quantum computing. Due to their ability to run on classical hardware, simulators allow researchers to study quantum systems without working with actual quantum hardware. Various optimizations have been invented and implemented in recent years. Among them, decision diagrams allow to efficiently store gates and quantum states in a designed data structure. We present a new simulation approach based on separating quantum states from each other such that every independent set of qubits is stored in a different decision diagram. Our research shows that this can potentially lead to slight improvements in regards to performance for some circuits.

Contents

Acknowledgments **iii**

Abstract **iv**

1 Introduction **1**

2 Background **2**
2.1 Quantum Computing 2
2.1.1 Qubits and Quantum States 2
2.1.2 Entanglement 3
2.1.3 Quantum State Representation 3
2.2 Related Work 4
2.2.1 The Munich Quantum Toolkit (MQT) 5
2.2.2 Previous Contributions to MQT DD Package 5
2.3 Decision Diagram Implementation 5

3 Entanglement-Split Simulator **8**
3.1 Gate Application 8
3.2 Conceptual idea 8
3.2.1 Reordering Decision Diagram 8
3.2.2 Variable Swapping 9
3.2.3 Sifting Algorithm 10
3.3 Implementation details 11
3.4 Test environment 11

4 Evaluation **13**
4.1 Experimental Setup 13
4.2 Results and Analysis 13
4.3 Discussion 14
4.4 Limitations 15

5 Future Work **16**
5.1 Automation of Entanglement-Split-Simulator Process 16

5.2 Decision Diagram Combination Strategies 16

List of Figures **17**

Listings **18**

Bibliography **19**

1 Introduction

Quantum simulations have an enormous impact on computational exploration and technological innovation. For instance, they help us understand protein folding which is essential for developing a cure for diseases such as Alzheimer's or Parkinson's. Another field of application is quantum cryptography[12]. Current cryptographic algorithms mainly use the following mathematical problems:

- Integer factorization problem
- Discrete logarithm problem
- Elliptic-curve discrete logarithm problem

Shor's algorithm allows us to solve these problems, assuming the number of supported qubits is sufficient[2]. This means that new quantum-safe cryptographic algorithms have to be developed. Researchers can leverage quantum simulators to gain valuable insights at a low cost.

The objective of this thesis is to introduce, analyze and evaluate new way of simulating a quantum circuit. The inspiration for the new approach was taken from [5], where certain advantages of keeping qubits separated from each other were demonstrated. We first dive into quantum computing by understanding basic gate operations and states relevant for our work. We continue by exploring decision diagrams from a theoretical perspective as well as by looking at the implementation details. This step is essential to understand advantages and disadvantages of our simulator. For now, our simulator only supports specific circuits due to a complex problem described in Chapter 3. However we hope to automate the combining process in the future to enable simulation of every possible circuit. Finally, we evaluate the results by comparing them on identical circuits run on different simulators and discuss possible further developments.

2 Background

In this chapter we provide information relevant for the following chapters. We start by exploring fundamentals of quantum computing, followed by an examination of the initial implementation of the MQT DDSIM tool.

2.1 Quantum Computing

Given the breadth of quantum computing, we will not delve into the entirety of its theory. Instead, we will focus on understanding terms that are important for comprehending the subsequent chapters.

2.1.1 Qubits and Quantum States

Qubits, short for quantum bits, form the fundamental units of information in quantum computing. Unlike classical bits that can exist in either a 0 or 1 state, qubits can exist in a superposition of both states simultaneously due to the principles of quantum mechanics. This means that a qubit can represent not just one, but an infinite amount of possible states between 0 and 1. For instance, if we consider using computational basis states, then we could get a qubit ψ in a superposition denoted as:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.1}$$

in Dirac notation. The coefficients α and β are complex numbers known as amplitudes [6]. According to Max Born, the probability of attaining a specific basis state after the measurement is equal to the modulus squared of the amplitude of the corresponding state. To be able to describe a quantum circuit we can measure a quantum state where each qubit collapses to one of the basis states with a certain probability. For our example in 2.1 we would get qubit ψ collapsed to $|0\rangle$ state with probability equal to $|\alpha|^2$ and collapsed to $|1\rangle$ with probability of $|\beta|^2$. Furthermore, the sum of the probabilities is equal to 1, from which the following equation is also true for amplitudes α and β:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.2}$$

$\|000\rangle$	$\|001\rangle$	$\|010\rangle$	$\|011\rangle$	$\|100\rangle$	$\|101\rangle$	$\|110\rangle$	$\|111\rangle$
$\frac{1}{\sqrt{10}}$	$\frac{-1}{\sqrt{10}}$	0	$\frac{2}{\sqrt{10}}$	0	0	0	$\frac{2i}{\sqrt{10}}$

Figure 1: Array representation of a quantum state with state $|q_2q_1q_0\rangle$ and corresponding normalized values

2.1.2 Entanglement

Entanglement is not existent in classical computers, however it is an important concept, especially for our work. Two qubits states are called entangled when the state of one qubit becomes dependent on the state of another qubit. For example, consider two entangled qubits: if one qubit is measured and found to be in the state $|0\rangle$, the other qubit instantly collapses into the corresponding entangled state, which could be $|1\rangle$, regardless of its physical separation.

2.1.3 Quantum State Representation

There exist multiple possibilities of representing quantum state during the simulation. One of them is the simple representation of an array as shown in Fig. 1. As you can see, each possible state has a corresponding value denoted below each possible outcome. More precisely, 2^n array entries are required for n qubits to represent a quantum state, following this straightforward approach, leading to exponential memory utilization.

Another possibility is to use Decision diagram (DD) that can reduce this exponential complexity by exploiting redundancies [7],[1]. It is important to highlight that exponential complexity continues to persist in worst-case scenarios for decision diagrams. If we compare representations Fig. 1 and Fig. 2, we will see the obvious structural differences. The amplitudes are encoded into edge weights in a DD and an empty edge weight can be translated to the weight value of 1. We can acquire the amplitude for a specific basis state by traversing the DD starting from the root edge, multiplying every edge weight on the path and making a decision every time a node is encountered. If the corresponding qubit is collapsed to $|0\rangle$ in our desired state, then we choose the left outgoing path, in case of $|1\rangle$ - we choose the right outgoing path. During the traversal, two types of terminal nodes are to be distinguished:

- Reaching a filled circle indicates an amplitude of zero.

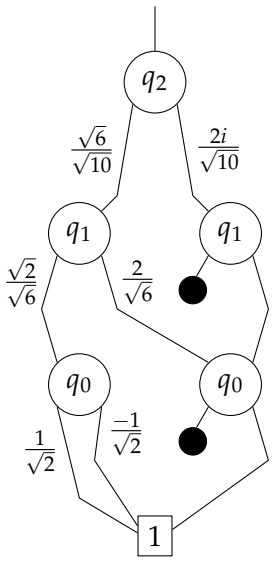


Figure 2: Decision Diagram representation of a quantum state $|q_2 q_1 q_0\rangle$

- Reaching the terminal "1" node indicates end of the traversal.

For example, yielding amplitude for a basis state $|011\rangle$ involves the following calculation:

$$1 \cdot \frac{\sqrt{6}}{\sqrt{10}} \cdot \frac{2}{\sqrt{6}} \cdot 1 = \frac{2}{\sqrt{10}} \tag{2.3}$$

We obtain the same result as from the array representation.

2.2 Related Work

The main benefit of using Decision Diagrams for Quantum Computing is an efficient representation of quantum states and gates. In recent years, significant progress in further development and optimization of those structures has been made. Since quantum computers are an emerging technology, most of the research and developments are performed on quantum simulators, on which we will focus below.

2.2.1 The Munich Quantum Toolkit (MQT)

MQT is a collection of quantum computing tools that were developed at the Chair for Design Automation at TUM [1]. For our purposes, MQT DD Package [17] is the most essential one for the research, however it might be also helpful to see how tools, such as MQT DDSIM [4] or MQT DDVIs [13], use the DD Package in their implementations. Every tool of the collection is open-source and is available on GitHub [2].

2.2.2 Previous Contributions to MQT DD Package

As mentioned above, various improvements were made and integrated in the project. Among them the Noise-aware Quantum Circuit Simulation [15] which takes noise effects into account that might happen during a simulation. Furthermore, efficient handling of complex numbers [16] was implemented. Finally, two other implementation concepts that improved the efficiency of the simulation include Unique Tables, Compute Tables[16].

- Unique Table is a chaining hash table which allows to remove redundancies by checking whether newly created node already exists. If that is the case, the resulting DD becomes more compact, improving the efficiency of further calculations and measurements.
- Compute Tables store results after computation is made (caching) that can be reused afterwards. During a simulation operations with same nodes might be particularly repeated, and thus the recomputation happens to be unnecessary.

Another important contribution is the implementation of the Hybrid Schrödinger-Feynman Simulator[3]. Although, this approach is different from ours, there exist certain similarities, in particular the splitting effect itself, where circuit is partitioned into blocks and the gates are decomposed using Schmidt decomposition.

2.3 Decision Diagram Implementation

Before we take a closer look at the implementation of our Entanglement-Split Simulator in Chapter 3, it is essential to understand the initial tool's structure since we reuse its underlying libraries. Depending on the decision diagram use case, we distinguish between matrices and vector nodes. For now, we are interested only in vector nodes

[1] `https://www.cda.cit.tum.de/research/quantum/mqt/`
[2] `https://github.com/cda-tum`

because matrix nodes represent gates, whereas vector nodes represent quantum states. Listing 1 shows that each node element contains the following attributes:

- Two outgoing edges. Each of them points to a corresponding node in the DD with a given weight[16].
- Pointer to the next node in the unique table.
- Reference count variable that counts the number of edges pointing to the node.
- Index variable representing qubit in the circuit.

```
1  struct vNode {
2     std::array<Edge<vNode>, 2> e{}; // edges out of this node
3     vNode* next{}; // used to link nodes in unique table
4     uint32_t ref{}; // reference count
5     int8_t v{}; // variable index (nonterminal) value (-1 for terminal)
6  }
7
8  struct Edge {
9     Node* p;
10    Complex w;
11 }
```

Listing 1: Node structure

To better understand the idea, we may now observe the circuit described in Fig. 3, assuming qubits are set to zero initially:

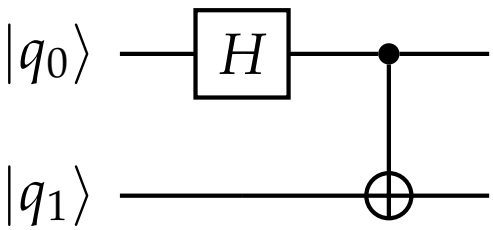


Figure 3: Circuit with H and CNOT gate

After applying both operations in the circuit, we achieve a decision tree of height three, including the terminal node, as shown in Fig. 4. Additionally, Fig. 4 provides

insights into the implementation details of the node and edge objects. Notably, the edge associated with q_1 showcases an imaginary weight of 0 and a real weight of 0.70710678118654757, corresponding, with a certain fidelity[14], to the theoretical value of $\frac{\sqrt{2}}{2}$. The edge also encapsulates the information of its destination node, in this case, the q_0 node situated on the right within the decision diagram. Subsequently traversing the decision diagram, the simulator would arrive at the node object that reveals the variable number designating a qubit in the circuit. This node features two outgoing edges with weights of 0 and 1. Remarkably, the diagram's design avoids the inclusion of additional terminal nodes (filled circles in the DD); instead, both edges converge on the same terminal node, therefore optimizing memory utilization. It is worth mentioning that the variable number -1 has been designed to exclusively denote a terminal node, signalling the conclusion of traversal within the decision diagram.

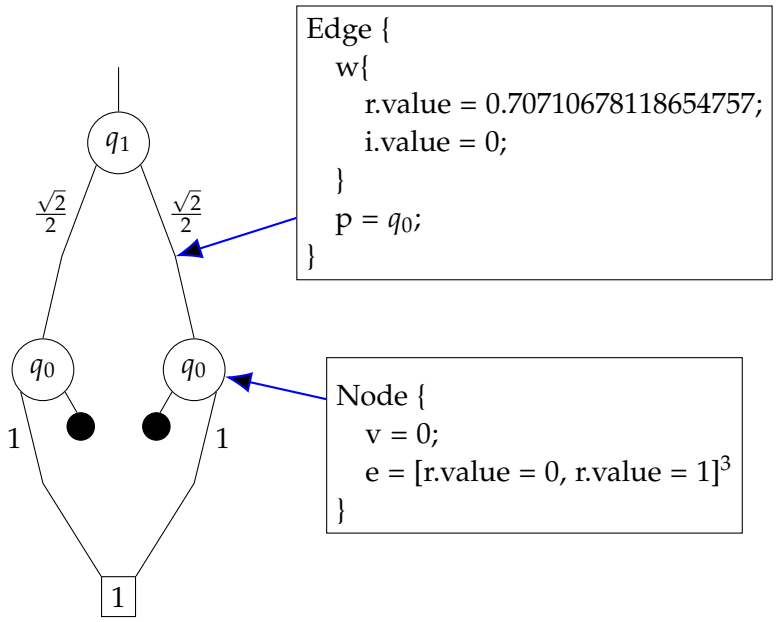


Figure 4: Resulting decision diagram of an example circuit

[1]For simplicity reasons only real values are shown in the array instead of the whole Edge objects

3 Entanglement-Split Simulator

In this chapter, we present the theoretical concept of our Entanglement-Split Simulator (ESS) and delve into the details of its implementation. Specifically, we introduce the on demand combination of decision diagrams, a strategy that enhances our simulator's efficiency while offering the flexibility to modify the order of qubit placement during the simulation.

3.1 Gate Application

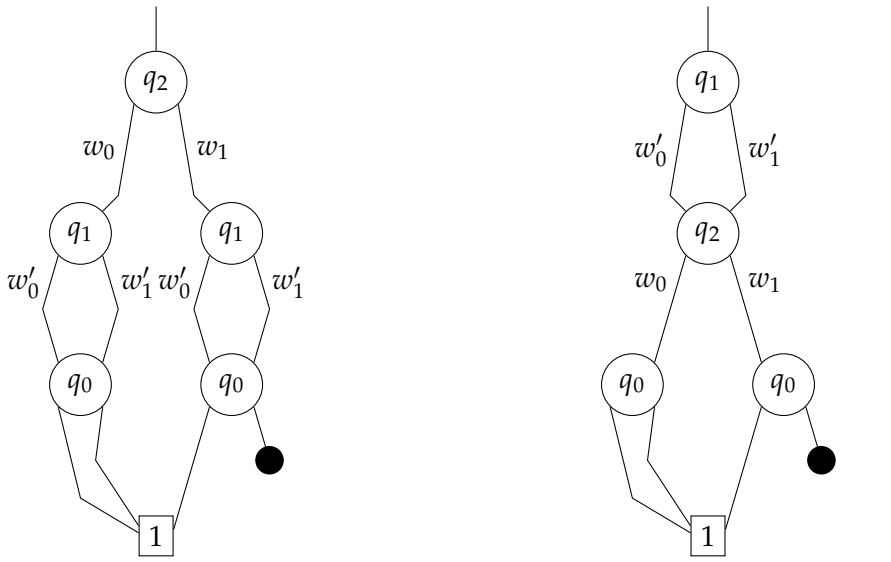
In the MQT DDSIM Tool, we utilize decision diagrams to model gate operations, and these diagrams can be transformed into matrices for the computation of updated quantum states. The separation of decision diagrams representing quantum states leads to the creation of matrices that are smaller in size, particularly evident in more extensive circuits. Consequently, this division significantly accelerates the process of applying operations.

3.2 Conceptual idea

Placing every qubit into one decision diagram at the start of the simulation has a clear disadvantage that the placing order[7] of the variables is predefined and can not be changed during the simulation. Instead, a sensible idea would be to try to make the ordering decision on demand. The key concept here is the iterative "merging" of pairs of decision diagrams, denoted as dd_1 and dd_2, just before the execution of an operation. This merging process is specifically applied when the operation involves qubits that are found both within dd_1 and dd_2. Computing kronecker product of two DDs is a fairly cheap operation in contrast with matrices; hence, $n-1$ kronecker products in total do not affect the simulation time drastically that can be seen in Chapter 4.

3.2.1 Reordering Decision Diagram

The number of nodes in a decision diagram can vary significantly based on the order of variables[7]. Determining an optimal variable order is however an NP-hard problem[9].



(a) DD with order q_2 - q_1 - q_0

(b) DD with order q_2 - q_2 - q_0

Figure 5: Changing variable order by swapping $q2$ with $q1$

Fig. 5 shows an example, in which reordering yields 20% reduction of non-terminal nodes. However, reordering causes new problems because of a limited floating-point accuracy on a classical computer. Exact representations [18] are extremely expensive, whereas adjusting weight values during reordering using floating-point values could result in the loss of precision, potentially leading to information loss or, more critically, node deletions due to collisions[18].

3.2.2 Variable Swapping

The determination of whether qubits should be merged from the top or the bottom presents a considerable challenge. Complexity lies in the potential application of sophisticated variable swapping techniques, as discussed in the context of dynamic variable ordering [11], particularly maintaining local complexity and logical consistency within the DD structure. For instance, one significant problem remains to efficiently identify nodes at a particular level. To tackle this, an innovative approach employs arrays of hash tables, each corresponding to a specific level within the diagram. This allows us to access nodes without traversing the entire structure from the roots. The hash tables further utilize bins to store nodes, facilitating an organized traversal process[11]. This approach might offer a helpful way to simplify decision-making involved in the merging process, ultimately leading to complexity reduction.

3.2.3 Sifting Algorithm

Another important concept is the DD minimization algorithm which we might take advantage of. When dealing with a DAG (Directed Acyclic Graph) representing the DD, various techniques have been explored to find optimal variable orderings. One such approach is the window permutation algorithm, where all levels in the DAG are marked. However, its effectiveness is limited, and when all levels are marked, the window permutation algorithm can no longer enhance the DAG size[10]. Due to the efficiency of swapping adjacent variables, the window permutation algorithm remains practical for relatively small values, typically around 4 or 5. However, research suggests that this algorithm has constraints in discovering favorable variable orders.

In response to these challenges, a new algorithm called the sifting algorithm has been proposed. This algorithm is centered around identifying the most suitable position for a variable within the DAG, under the assumption that all other variables remain fixed. Given n variables in the DAG (excluding the constant level that always resides at the bottom), there exist n potential positions for a variable, including its current location. Among these n positions, the goal of the sifting algorithm is to find the position that minimizes the DAG's size. Although it would be ideal to determine the best position for a variable with a low-complexity analysis of the DD, making this idea work in practice is difficult. Consequently, the sifting algorithm adopts a brute-force enumeration approach to ensure the optimal variable position[10]. The procedure involves swapping the variable with its successor[8] until it becomes the penultimate variable in the DAG—sifting it down to the bottom. Subsequently, the variable is swapped with its predecessor until it reaches the top position in the DAG—sifting it to the top. Throughout this process, the best DAG size observed is recorded, and the variable's position is then adjusted to its optimal location. The sifting algorithm's execution follows a structured process. First, the variables are sorted based on the number of nodes at each level in the graph, with the order determined by decreasing size. Then, each variable is moved to its locally optimal position, while assuming that all other variables remain fixed. Notably, each variable is only moved once in this process, although the algorithm has the potential for iterative convergence.

One of the significant advantages of the sift algorithm is its capacity to facilitate substantial movement of variables within the ordering. It's important to note that the size of the DAG can increase significantly after the initial variable swaps and eventually decrease below the starting point. This allows for uphill moves, where the acceptance of a sequence of pairwise swaps is based on the best position identified, regardless of any temporary increase in the intermediate DAG size. In contrast, the window permutation algorithm can encounter challenges when multiple moves are required to shift a variable a significant distance, which might be hindered by intermediate uphill

moves. In terms of computational complexity, the sift algorithm necessitates $O(n^2)$ swaps of adjacent levels in the DAG. Additionally, each of these variable swaps incurs a complexity proportional to the DAG's width. To control the worst-case complexity, the algorithm stops the search in a particular direction if the DAG's size grows to double its original size. This combination of heuristics makes the sifting algorithm valuable for refining variable orders and optimizing decision diagram structures, which underlines the importance of choosing an appropriate variable swapping algorithm.

3.3 Implementation details

To start with, to reuse the existing DD Package reasonably, we use so-called *slices* having a structure described in Fig. 2. With its help, we define *start* and *end* qubit variables and root edge, whereas *nqubits* specifies the number of qubits located in the slice.

```
1 class Slice {
2    int8_t start;
3    int8_t end;
4    int8_t nqubits;
5    Edge edge{};
```

Listing 2: Slice implementation

Furthermore, keeping track of which slice contains which qubit requires an additional data structure called $union - table$. We were inspired by the idea provided in [5] and decided to use an array where each entry in position i defines which slice contains qubit i. Both slices and the designed data structure have to be updated after each operation to display the correct state of the quantum system. The pointer to the root edge of the final decision diagram is stored in every entry of the array after simulation, so it can be simply retrieved, for example, by accessing the first entry of the resulting array.

Before applying each operation we first seek for *slices* with intersecting target or control qubits. Then, we combine such *slices*, if needed, and update $union - table$, followed by application of the corresponding operation.

3.4 Test environment

To ensure correctness of the simulator, we performed several tests on different quantum circuits and compared the resulting amplitudes with the expected ones. In the Listing 3 you can see an example quantum circuit described in Fig. 3. We simulate the circuit one time expecting amplitudes after measurement $|00\rangle$ and $|11\rangle$ be equally probable with $P = 0.5$ each. We also define delta parameter, which allows certain deviations

```
[ RUN      ] EntanglementSplitSimulatorDDSIM.TrivialTest
resultAmp[00] = 61
resultAmp[11] = 67
[       OK ] EntanglementSplitSimulatorDDSIM.TrivialTest
```

Figure 6: Console output of the example test run

from the expected result. Since we define 128 measurement shots, we expect each of the two possible states to be measured 64 times on average. The resulting console output can be seen in Fig. 6.

```
1  TEST(EntanglementSplitSimulatorDDSIM, TrivialTest) {
2     auto quantumComputation = [] {
3         auto qc = std::make_unique<qc::QuantumComputation>(2);
4         qc->h(0); // 1| h q[0];
5         qc->x(1, {0_pc}); // 2| cx q[0], q[1];
6         return qc;
7     };
8
9     EntanglementSplitSimulator ddsim(quantumComputation());
10
11    auto resultAmp = ddsim.simulate(128);
12    for (const auto& entry: resultAmp) {
13        std::cout << "resultAmp[" << entry.first << "] = " << entry.second << "\n";
14    }
15
16    ASSERT_EQ(resultAmp.size(), 2);
17    auto it = resultAmp.find("00");
18    ASSERT_TRUE(it != resultAmp.end());
19    EXPECT_NEAR(static_cast<double>(it->second), 64, 32);
20    it = resultAmp.find("11");
21    ASSERT_TRUE(it != resultAmp.end());
22    EXPECT_NEAR(static_cast<double>(it->second), 64, 32);
```

Listing 3: Test example

4 Evaluation

This chapter presents the evaluation of the proposed approaches, namely the Entanglement-Split Simulator (ESS) and the Circuit Simulator (CS), in the context of quantum circuit simulation. Circuit Simulator is a "naive" Simulator with minimal optimizations, whereas Entanglement-Split Simulator was described in Chapter 3. The performance of these methods is assessed using a set of representative quantum circuit configurations: $grover_7$, $grover_{15}$, and $grover_{19}$. The configurations were derived from their respective qasm files, namely *grover*_19.*qasm*, *grover*_15.*qasm* and *grover*_7.*qasm*, which were generously provided by the Chair for Design Automation. Notably, these configurations underwent an essential modification in their variable order to accommodate the operation of the Entanglement-Split Simulator. This adjustment was necessary due to ESS's specific implementation requirements, ensuring that the circuits could be successfully simulated using this novel approach. The evaluation aims to provide insights into the efficiency and effectiveness of these simulation techniques in terms of their gate processing rates.

4.1 Experimental Setup

To evaluate the performance of the ESS and CS methods, a series of experiments were conducted using the aforementioned quantum circuit configurations. All the experiments were executed on a laptop with the following specifications:

- CPU: 3.5GHz Dual-Core Intel Core i7
- Memory: RAM: 16 GB 2133 MHz LPDDR3

We conducted each test exactly ten times and calculated the average time across the trials. The number of gates processed per second were recorded as the metrics for comparison, with a longer bar indicating greater performance.

4.2 Results and Analysis

The results of the evaluation are presented using a horizontal bar chart, as shown in Fig. 7. This chart visually illustrates the gate processing rates achieved by the ESS and

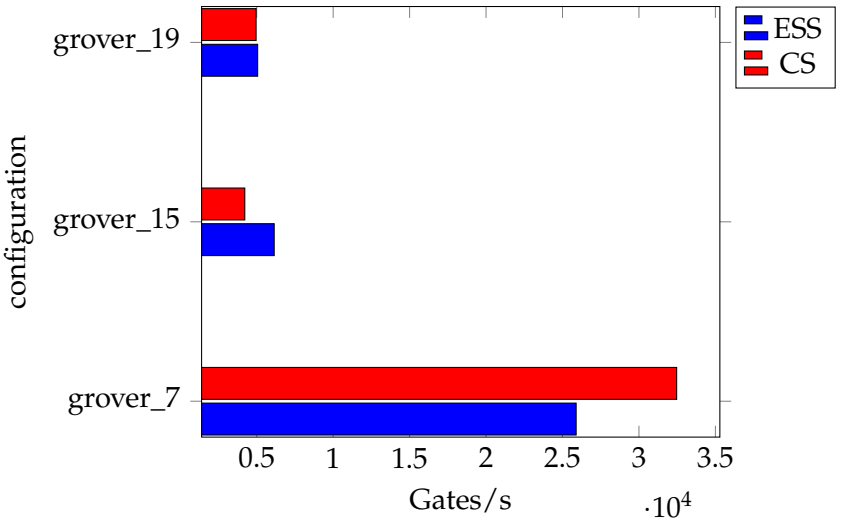


Figure 7: Gate Processing Rates Comparison for Entanglement-Split Simulator (ESS) and Circuit Simulator (CS Methods.

CS methods for each of the three circuit configurations.

The analysis of the results reveals intriguing insights. The ESS method showcases a distinct trend in gate processing rates across the various circuit configurations. Notably, for the $grover_7$ configuration, the ESS method achieved a commendable gate processing rate of 25,892 gates per second. However, this rate decreased to 6,143 and 5,062 gates per second for the $grover_{15}$ and $grover_{19}$ configurations, respectively. In contrast, the CS method demonstrated a different pattern. It achieved notably higher gate processing rates for the $grover_7$ configuration (32,467 gates per second) and the $grover_{19}$ configuration (4,960 gates per second), while achieving a rate of 4,220 gates per second for the $grover_{15}$ configuration.

4.3 Discussion

The observed variations in gate processing rates can be attributed to the inherent characteristics of the ESS and CS methods. Both simulators exhibited notably higher gate processing rates when applied to the $grover_7$ configuration. This phenomenon can potentially be attributed to the exponential growth of decision diagrams. As the number of qubits increases, along with a fixed simulation time, the gate operations in configurations like $grover_{19}$ and $grover_{15}$ experience substantial delays due to the exponential growth of decision diagrams. In circuits with a substantially larger number of qubits, the depth of the decision diagram increases. This effect contributes to

prolonged execution times for operations in the circuit. For comparison, $grover_{19}$, $grover_{15}$, and $grover_7$ require 35, 27, and 11 qubits respectively, with corresponding differences in the depth of the decision diagrams. However, it is important to note that our assessment of ESS was conducted on a single variant out of the 2^n (where n represents the number of qubits) possible variants, where we attached DD with a root node having higher variable number from the bottom to the DD with a root node having lower variable number, so that the resulting root node possesses higher variable number. We assume that proceeding differently for same configurations might result in a completely different outcome. Furthermore, it is worth considering that for more complex configurations, ESS might yield significantly improved results compared to CS due to decreasing gate processing rates, where ESS can take a bigger advantage of splitting decision diagrams.

4.4 Limitations

Unfortunately, the current capabilities of ESS are constrained to circuits in which the combination of decision diagrams is limited to those continuous in order. To ensure adherence to this constraint, we initially performed a permutation of qubits on the provided qasm files. This process should be automated in subsequent developments.

5 Future Work

This chapter explains possible future exploration that would move the present research forward, and open new potential paths to enhance the capabilities and efficiency of the discussed simulation approach. The forthcoming sections draw the lines of key directions for subsequent studies while emphasizing the importance of comprehensive assessment and optimization.

5.1 Automation of Entanglement-Split-Simulator Process

An important objective for future investigations is the automation of the qubit permutation process. During our measurements, we needed a manual preprocessing step, wherein qubit permutations were conducted before the initiation of simulations. Achieving the automation of this process would not only make simulation procedures more efficient, but also broaden the scope of applicable circuit configurations, improving the overall versatility of the simulation process.

5.2 Decision Diagram Combination Strategies

A fundamental benefit of simulation is the possession of insights that can influence the approach. Analyzing better how decision diagrams are optimally combined [7], such as from the top or bottom, holds the potential to uncover strategies that optimize simulation efficiency, specifically, in terms of reducing the number of nodes within a decision diagram. Delaying the combination of decision diagrams not only reduces redundancy but also provides the simulator with valuable insights into potential optimal combination strategies, offering the possibility of a better performance by choosing whether to combine qubits from the top or bottom. This change could minimize complexity, thus resulting in a more scalable simulation process.

List of Figures

1 Array representation of a quantum state with state $|q_2q_1q_0\rangle$ and corresponding normalized values 3
2 Decision Diagram representation of a quantum state $|q_2q_1q_0\rangle$ 4
3 Circuit with H and CNOT gate 6
4 Resulting decision diagram of an example circuit 7

5 Changing variable order by swapping $q2$ with $q1$ 9
6 Console output of the example test run 12

7 Gate Processing Rates Comparison for Entanglement-Split Simulator (ESS) and Circuit Simulator (CS Methods. 14

Listings

1 Node structure 6

2 Slice implementation 11
3 Test example 12

Bibliography

[1] A. Abdollahi and M. Pedram. "Analysis and Synthesis of Quantum Circuits by Using Quantum Decision Diagrams." In: *Proceedings of the Design Automation & Test in Europe Conference*. Vol. 1. 2006, pp. 1–6. DOI: 10.1109/DATE.2006.244176.

[2] D. J. Bernstein. "Introduction to post-quantum cryptography." In: *Post-Quantum Cryptography*. Ed. by D. J. Bernstein, J. Buchmann, and E. Dahmen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–14. ISBN: 978-3-540-88702-7. DOI: 10.1007/978-3-540-88702-7_1.

[3] L. Burgholzer, H. Bauer, and R. Wille. "Hybrid Schrödinger-Feynman Simulation of Quantum Circuits With Decision Diagrams." In: *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, Oct. 2021. DOI: 10.1109/qce52317.2021.00037.

[4] Chair for Design Automation at TUM. *A quantum circuit simulator based on decision diagrams written in C++*.

[5] Y. Chen and Y. Stade. "Quantum Constant Propagation." May 2023.

[6] J. D. Hidary. *Quantum Computing: An Applied Approach*. English. Springer International Publishing AG, 2019. DOI: 10.1007/978-3-030-23922-0.

[7] S. Hillmich et al. "Reordering Decision Diagrams for Quantum Computing Is Harder Than You Might Think." In: *Reversible Computation*. Ed. by C. A. Mezzina and K. Podlaski. Cham: Springer International Publishing, 2022, pp. 93–107. ISBN: 978-3-031-09005-9. DOI: 10.1007/978-3-031-09005-9_7.

[8] C. Jiang et al. "Variable Reordering in Binary Decision Diagrams." In: *26th International Workshop on Logic & Synthesis* ().

[9] C.-C. Lam, P. Sadayappan, and R. Wenger. "On Optimizing a Class of Multi-Dimensional Loops with Reductions for Parallel Execution." In: *Parallel Process. Lett.* 7 (1997), pp. 157–168.

[10] C. Meinel, F. Somenzi, and T. Theobald. "Linear sifting of decision diagrams and its application in synthesis." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 19.5 (2000), pp. 521–533. DOI: 10.1109/43.845077.

[11] R. Rudell. "Dynamic variable ordering for ordered binary decision diagrams." In: *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*. 1993, pp. 42–47. DOI: 10.1109/ICCAD.1993.580029.

[12] S. Wang, R. Matthew, and A. Amjad. "Quantum Cryptography and Simulation: Tools and Techniques." In: (Feb. 2020), pp. 36–41. DOI: 10.1145/3377644.3377671.

[13] R. Wille, L. Burgholzer, and M. Artner. "Visualizing decision diagrams for quantum computing." In: *Design, Automation and Test in Europe*. 2021. DOI: 10.23919/DATE51398.2021.9474236.

[14] R. Wille, S. Hillmich, and L. Burgholzer. "Decision Diagrams for Quantum Computing." In: *Design Automation of Quantum Computers*. Springer International Publishing, Aug. 2022, pp. 1–23. DOI: 10.1007/978-3-031-15699-1_1.

[15] A. Zulehner, J. Fuß, and R. Wille. "Noise-aware Quantum Circuit Simulation With Decision Diagrams." In: *Transactions on CAD of Integrated Circuits and Systems (TCAD)* (2022). DOI: 10.1109/TCAD.2022.3182628.

[16] A. Zulehner, S. Hillmich, and R. Wille. "How to Efficiently Handle Complex Values? Implementing Decision Diagrams for Quantum Computing." In: *International Conference on Computer Aided Design (ICCAD)* (2019). DOI: 10.48550/arXiv.1911.12691.

[17] A. Zulehner and R. Wille. "Advanced Simulation of Quantum Computations." In: *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems (TCAD)* (2018). DOI: 10.48550/arXiv.1707.00865.

[18] A. Zulehner et al. "Accuracy and Compactness in Decision Diagrams for Quantum Computation." In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2019, pp. 280–283. DOI: 10.23919/DATE.2019.8715040.