RWTH Aachen

Bachelor thesis
in Computer Science

Blackbox and Whitebox Algorithms for Efficient Parallel Quantum Computation

Author
Sebastian Issel

Supervisors
Prof. Dr. David DiVincenzo
Prof. Dr. Erich Grädel

Submitted to the
Faculty of Mathematics, Computer Science and Natural Sciences

March 2020

The first part of this paper contains an introduction to quantum computer science, at least to all parts needed for the remainder of the paper. It's written from a computer scientific point of view, and contains barely any physics. Only the postulates of quantum mechanics and some general statements, to reason about physical quantum computers, are used. Next, a generalization of Simon's problem, the inspiration of Shor's algorithm, is presented. A wider range of oracles turn out to be solvable with a similar solution. Only the classical part needs to be adjusted. The run-time and number of queries to the oracle stay the same. After that, it's shown that a tensor, or higher dimensional matrix, can hide a linear function. This is used to define Whitebox-versions of known Blackbox-problems. The tensor is given as the input, and the problems revolve around finding the linear function. Different versions are shown. One of them is very likely to be faster solvable on a quantum computer than possible on a classical one. The run-time is constant on the quantum computer, while the classical one needs likely at least logarithmic time. This is because it is relatively similar to a problem that is proven to be harder to solve on a classical computer. A constant quantum solution for this problem is presented. For small instances of the problem, an efficient classical solution is presented.

Contents

1 Introduction

The theoretical workings of quantum computation are unknown for most computer scientists. Most are aware that an algorithm to factor numbers efficiently is known, and that they use qubits, whatever they may be. To allow a broader audience, the needed basics for a computer scientist to understand this paper are presented.

The base for quantum computation are the qubit and all the transformation that are possible on it. Qubits obey the rules of quantum mechanics. They can therefore interact in ways simply not possible in a classical machine. Quantum mechanics is not needed for theoretical quantum computer science, only the possible interactions and transformations have to be known. The only given physics are some postulates of quantum mechanics and implications for actual quantum computers.

1.1 A New Kind of Memory

A qubit could be any two-state quantum system. Two-state meaning that a measurement can return one of two different results. Quantum mechanics dictates that such a system is described by a two-dimensional complex Hilbert-space. Let's call two orthogonal states of this space $|0\rangle$ and $|1\rangle$. Those states will be referred to as base states. Every other state is called entangled. Because of their orthogonality, they will have a special property, they can be distinguished with certainty via a measurement. It could seem odd for this to be reserved to orthogonal states, therefore Section §1.3 will focus on measurements and talk more about why this is. The base states are obviously not unique, since they are just the orthogonal bases of a two-dimensional Hilbert-space. Which one is chosen depends on the physical device, how easy the transformations can be implemented with them, and other properties. The state of a qubit can be described by a unitary vector from the Hilbert-space. Instead of just $|0\rangle$ or $|1\rangle$, the general state of a qubit is therefore described by

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

for $\alpha, \beta \in \mathbb{C}$ where

$$|\alpha|^2 + |\beta|^2 = 1$$

holds.[1]

While an individual qubit is described by a two-dimensional vector space, the dimensions don't simply add up for multiple. Quantum mechanics dictates that the full system is described by the tensor product over all its subsystems. It follows that a system with q

[1] States and their representation explained in [15, pp. 17-19].

qubits is described by a 2^q-dimensional vector. As an example, a two qubit system can be described as the sum over the four base sates

$$|\psi\rangle = \alpha\,|00\rangle + \beta\,|01\rangle + \gamma\,|10\rangle + \delta\,|11\rangle = \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix}$$

with the normalization condition

$$|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$$

The state of three qubits has eight dimensions, four have 16 dimensions, and so on. The reason a physical system should behave in this way, is simply because of quantum mechanics. It's postulates dictate that an isolated two-state system is described in precisely this way, and multiple systems are combined with the tensor product. The idea of quantum computers is to exploit these non-classical behaviors for new computational power.[2]

There are two important notes to the phase of a state. First, the state described by $|\psi\rangle$ is precisely the same as $e^{i\varphi}\,|\psi\rangle$, for any $\varphi \in \mathbb{R}$. This is called a global phase. It can be thought of as another normalization factor. Second, the phase is important, if it differs between different bases, a relative phase. This means $|1\rangle \equiv -\,|1\rangle$ but $|0\rangle + |1\rangle \not\equiv |0\rangle - |1\rangle$. In the example, the states $|0\rangle + |1\rangle$ and $|0\rangle - |1\rangle$ are not normalized. The normalized states would be $\frac{1}{\sqrt{2}}\,(|0\rangle + |1\rangle)$ and $\frac{1}{\sqrt{2}}\,(|0\rangle - |1\rangle)$. Since it's unambiguous what states were meant, this can be dropped for convenience.[3]

$|0\rangle$ and $|1\rangle$ is of course not the only possible base to write the states in. The states could be written along any valid base. Sometimes $|+\rangle := \frac{1}{\sqrt{2}}\,(|0\rangle + |1\rangle)$ and $|-\rangle := \frac{1}{\sqrt{2}}\,(|0\rangle - |1\rangle)$ is used. Those states are as far as possible from a classical one, and have a special meaning, as will be seen in section 1.2.1.[4]

1.2 Reversible Transformations

To be able to calculate with this new memory, its state has to be manipulable. If all possible computational power is to be exploited, it has to be known what manipulations are possible. Quantum mechanics already has the answer to this. The physically possible transformations are linear and reversible. Since only the transformations from one normalized representatives to another are considered, precisely the unitary transformations are possible.[5]

The question arises if a quantum computer is even able to perform all classical operations. Many of the common logical gates are not reversible, simply because they

[2] Composing of systems introduced in [12, pp. 45-46].
[3] Global and relative phase explained in [12, pp. 40-41].
[4] Different bases explained in [16, p. 22].
[5] Presented in [16, pp. 17-22].

have a different amount of in- and outputs, and are therefore not possible to execute as a transformation. As a result, a classical computer is able to perform transformations, like AND, that are impossible for a quantum computer. However, there are easy workarounds to this problem. Instead of the non-reversible transformation, a reversible version is executed. Given a boolean function $f\colon \{0,1\}^n \to \{0,1\}^m$, a quantum computer can always perform a reversible version of that function. There is a simple way to achieve this. While leaving the input alone, the result is written into another dedicated register. The actually performed partial function could look like this:

$$\hat{f}\colon \{0,1\}^{n+m} \to \{0,1\}^{m+n}, (x_1 \dots x_n 0 \dots 0) \mapsto (x_1 \dots x_n f(x_1 \dots x_n)_1 \dots f(x_1 \dots x_n)_m)$$

One way to write this in terms of a quantum computers is

$$|x\rangle |0\rangle \xrightarrow{\hat{f}} |x\rangle |f(x)\rangle$$

To get a function that is defined for all inputs, often

$$|x\rangle |y\rangle \xrightarrow{\hat{f}} |x\rangle |y \oplus f(x)\rangle$$

is used. This is automatically reversible. By forcing the register y to be 0, f can be performed. To actually realize the function in a reversible way, it may be necessary that intermediate results are stored in memory. This means access to temporary memory is also needed, so-called ancilla qubits. It can be assumed that they are given in state 0, and can be returned once they are back in state 0. Since all transformations are reversible, every result can be uncomputed using the inverse transformation. If the result is first "copied", see Section §1.7, it's possible to keep the result, while resetting all used memory to 0. This allows for all ancilla bits to be returned, making them reusable. This is demonstrated on an example in section 1.2.4.[6]

1.2.1 Common Gates

Quantum gates can be expressed as unitary matrices. This is however not practical for representing an algorithm. If multiple qubits are involved, the matrices get huge very quickly, since they grow exponentially with the number of qubits. It also may not contain the structure of the algorithm in a recognizable way. For those reason the circuit model is often used. Here, the qubits are represented with wires, and the gates are written with symbols. A gate is often represented by a box containing its name, some have special symbols however. The qubits are written on the left and follow their line, leading into the gates. On the right side, the output emerges.[7]

If a gate U is applied on the state $|\psi\rangle$, the state after the transformation will be $U|\psi\rangle$. As an example let

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

[6] Reversible computation explained in [12, pp. 12-15].
[7] Circuit model introduced in [12, pp. 71-73].

some state and

$$U = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix}$$

be some gate. If the gate is applied to $|\psi\rangle$, then the resulting state is given by

$$U|\psi\rangle = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = (\alpha u_{11} + \beta u_{12})|0\rangle + (\alpha u_{21} + \beta u_{22})|1\rangle$$

Transformations can be applied in the same way for multiple qubits.[8]

Very important gates are now presented, with their symbols in the circuit model shown in Figure 1.1. Starting with one qubit operations, a very important gate is the Hadamard H.

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

It has no classical analogy and, in some sense, captures precisely what a quantum computer can do more than a classical one, as will be seen in section 1.2.3. It also holds that $H|0\rangle = |+\rangle$ and $H|1\rangle = |-\rangle$.[9]

Next, there are the phase-gate S and the $\pi/8$-gate T.

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \quad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} \equiv \begin{pmatrix} e^{-i\pi/8} & 0 \\ 0 & e^{i\pi/8} \end{pmatrix}$$

The second representation of T is where its name comes from. The representations only differ in a global phase, and are therefore the same transformation, just like $e^0 = e^{i2\pi}$. Their importance will become clearer in section 1.2.3. As a note, one is the square of the other $S = T^2$.[10]

The last presented one-qubit gates have their roots in physics, but represent important logical operations, the Pauli matrices.

$$\sigma_0 = I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \sigma_1 = X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \sigma_2 = Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \sigma_3 = Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

σ_0 is sometimes not counted, but the identity matrix corresponds to saving memory, keeping it unchanged. The others correspond to negation in different contexts. X is precisely the *NOT* operation and Z is the negation for the phase. Their close relation can be seen from $X = HZH$ and $XZ \equiv Y$. As another note $Z = S^2$.[11]

It turns out that for every gate U, there is also a controlled version CU.

$$CU = \begin{pmatrix} I & 0 \\ 0 & U \end{pmatrix}$$

Here the gate performs only if the control bit is 1, and acts as the identity otherwise. This can also be reversed, such that the gate only acts when the control is 0. One can be

[8] Linear algebra formulation shown in [12, pp. 8-12].
[9] Hadamard and representation in [5, pp. 122-123].
[10] Gates with matrix representation from [16, p. 174].
[11] Pauli matrices with further explanation and alternative representation by [5, pp. 121-122].

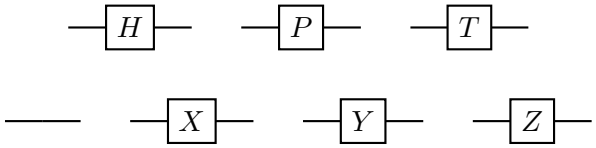
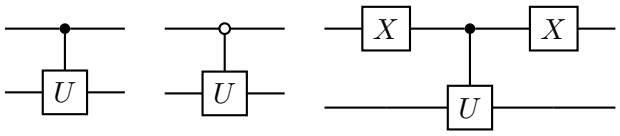


Figure 1.1: Pauli-matrices and other important one qubit gates in the circuit model.

realized from the other, simply by using an X before and after the control. In Figures 1.2 this is shown. Figure 1.2a shows the symbol for CU, while Figure 1.2b and Figure 1.2c depict the inverted version, and a way to change the control. A black dot is used for gates that apply if the control is one, and a white one if it applies on zero.[12]



(a) Control on 1. (b) Control on 0. (c) Changing the control.

Figure 1.2: Controlled versions of U.

Moving on to the classical part of calculations. Boolean functions are covered by

$$CX = CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

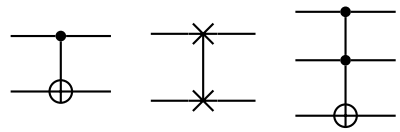
and the Toffoli-gate.

$$CCX = Toff = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

When starting with a base state, they only produce base states. This means that circuits only using those operations, are simple to simulate for a classical computer. CX stands for the controlled X and CCX for the double controlled X. In Figures 1.3 the dots

[12]Controlled gates with the general construction in [16, pp. 177-185].

correspond to the control bit and the other symbol to the target, where the negation is applied, or the swap performed.[13]



(a) *CNOT* gate. (b) Swap gate. (c) Toffoli gate.

Figure 1.3: Symbols for the classical part.

1.2.2 How Gates Combine

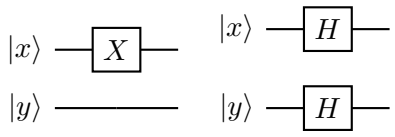
The matrix for the transformation of a full system will now be calculated. For a gate applied to the first qubit U,

$$U = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix}$$

and V some gate applied to the next q qubits, their combined transformation is given by $U \otimes V$.

$$U \otimes V = \begin{pmatrix} u_{11}V & u_{12}V \\ u_{21}V & u_{22}V \end{pmatrix}$$

Here the V is a $2^q \times 2^q$ unitary matrix. Let's look at the circuit from Figure 1.4a. The



(a) Only one Gate. (b) Only H gates.

Figure 1.4: Two example circuits.

NOT or X is applied to register x and nothing, meaning the identity, to y. Therefore, the transformation to the full system is $X \otimes I$. This results in

$$X \otimes I = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

[13] *NOT* and Toffoli presented in [5, pp. 106-108]. *SWAP* with an implementation given in [16, p. 23].

as the transformation of the circuit. If U is not a single qubit transformation, but acts on p qubits, the resulting matrix $U \otimes V$ is constructed in a similar way.

$$U \otimes V = \begin{pmatrix} u_{11}V & \cdots & u_{12^p}V \\ \vdots & \ddots & \vdots \\ u_{2^p1}V & \cdots & u_{2^p2^p}V \end{pmatrix}$$

Let's use the circuit from Figure 1.4b as another example. To both x and y, the Hadamard H is applied. Therefore, $H \otimes H$ is applied to the full system.

$$H \otimes H = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

This gate can also be written as $H^{\otimes 2}$ or in general $H^{\otimes n} := \bigotimes_{k=1}^{n} H$.[14]

1.2.3 Universal Sets of Gates

A completely general gate, acting on q qubits, can be represented by a $2^q \times 2^q$ unitary matrix U. Using Givens rotation, it is possible to factor the matrix into gates that only act on two qubits. This factorization needs some control logic that can also be realized with two-qubit gates. Two-qubit gates can in turn be build, using only one-qubit gates and *CNOT*. This means, the set containing just *CNOT* and all unitary 2×2 matrices is universal for quantum computation. An obvious problem is that this set is not finite and not even countable. Since uncountable many gates are supposed to be build-able with finitely many steps, this is unavoidable.[15]

Physical quantum computers won't be able to execute the gates perfectly. With this in mind, a set that's able to approximate every gate, up to some error, should be sufficient for any achievable computation. If such an imprecision is allowed, a finite set is sufficient to be universal. It turns out that $\{CNOT, H, T\}$ can approximate every gate arbitrary close. However, this has an overhead that is up to exponential in the allowed error. The allowed error could be chosen in the range of the unavoidable operation error of the physical device. This would give the least possible overhead for the maximal achievable precision. This construction overhead turns out to also be unavoidable, and the best known construction is very close to a theoretical lower bound.[16]

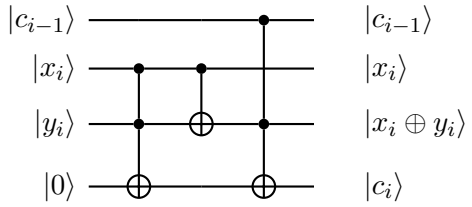
There are other universal sets. The one presented shows a separation of the classical and quantum part, in some sense. The Toffoli gate is universal for the reversible boolean functions, if one allows ancilla bits. Therefore, a quantum computer that can perform the Toffoli gate, and has access to ancilla qubits, can evaluate all classical reversible functions. At the same time, if the calculation only starts in base states, the Toffoli gate can only produce base states. Together, this means that the Toffoli gate covers exactly

[14] Tensor rules explained by [12, pp. 33-35].
[15] Constructive proof of universality in [16, pp. 188-194].
[16] Construction of finite set and its overhead in [16, pp. 194-200].

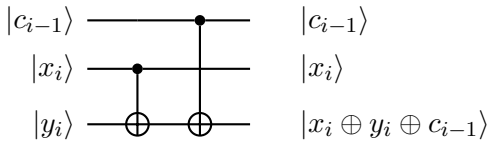
the classical part of a quantum computer. It turns out that only one more gate is needed to complete a universal set, the Hadamard H. Since the Hadamard gate has no classical equivalent and completes the set, it captures precisely what a quantum computer can do more than a classical one. It's important to note that the set $\{CCX, H\}$ is only universal to a more relaxed definition. All possible calculations can be simulated, but not every state can be realized. Besides ancilla bits, a special encoding is used. In this encoding, every logical qubit is divided into a real and imaginary part. For instance, the gate CP can not be performed, but it's possible to perform CCZ. By setting the target of CCZ on the imaginary part of the logical target and both controls on the real and imaginary part of the logical control, the desired gate is realized.[17]

1.2.4 Quantum Adder

As an example for a quantum algorithm and some concepts of reversible calculation, like uncomputing, the quantum adder is presented. Two sub-circuits will be used. One is called Carry and the other SUM, depicted in Figure 1.5a and Figure 1.5b. Carry



(a) Circuit Carry.



(b) Circuit SUM.

Figure 1.5: Needed Circuits for the adder.

calculates the carry for the next bit, while SUM simply adds two bits and their carry modulo two. Carry has the side effect of changing register y. Two different ways are used to handle this problem. For the highest bit, it's simply reversed. For the lower bits, the reversion is part of uncomputing the now unneeded carries. Figure 1.6 shows a circuit implementing the addition for two qubits. The two qubit inputs, $|x\rangle$ and $|y\rangle$, are given, as are two ancilla bits for the intermediate carries, and one extra output bit for the highest carry.[18]

For the highest bit, a single $CNOT$ is added to adjust for the side effect of the Carry circuit. The others, in this case only one, get a reversed Carry circuit to uncompute the first one, reverting the ancilla bit to 0. While the first ancilla bit is not needed in this case, it allows a very simple way to turn the adder into a subtractor. This is done in the same way as in the classical case, by using the two's complement. To realize this, the number to be subtracted is inverted, and one added, by flipping the first ancilla bit.[19]

[17] A proof for the universality and the definition of the special universality are shown in [3].
[18] Circuits for the adder from [8].
[19] Uncomputing in greater detail is given by [1, pp. 133-149].

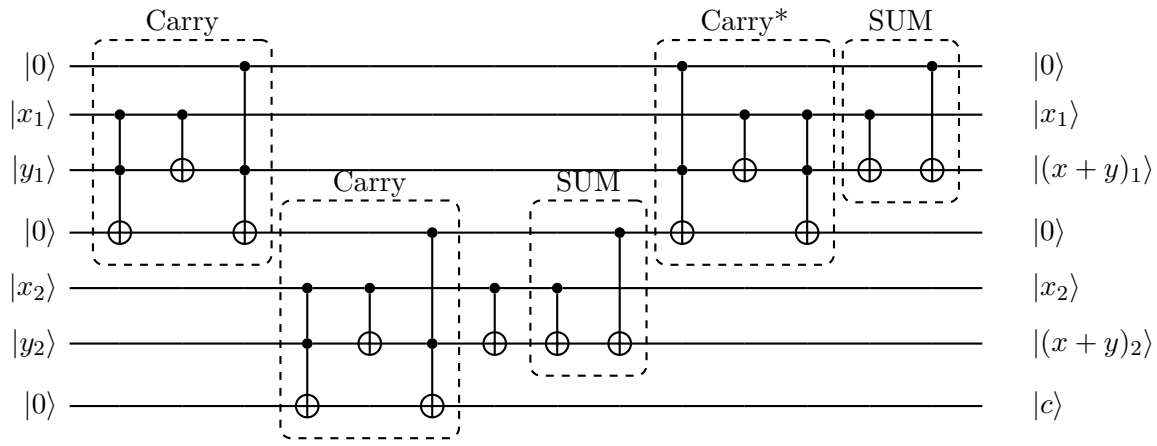


Figure 1.6: Circuit for a two bit quantum adder.

1.3 Measurements

A measurement is another needed operation for quantum computation. A general state can not be observed, only the outcome of a measurement can. The symbol for a measurement of one qubit in the circuit model is shown in Figure 1.7.[20]

Figure 1.7: Symbol for measurement with classical output.

It's sufficient to be able to measure in the $|0\rangle$ and $|1\rangle$ base. This base is also called Z base, because $|0\rangle$ and $|1\rangle$ are the eigenvectors of the Pauli Z matrix. Other orthogonal bases can be simulated, using a transformation into the base, measuring, and transforming back.[21]

Given a qubit in state $|\psi\rangle$.

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

If the qubit is measured in the Z base, then the result will be 0 with portability $|\alpha|^2$ and 1 with probability $|\beta|^2$. The result of a measurement is returned as a classical bit. After the measurement, the qubit will be in the measured state. This seems to contradict the postulate that only unitary transformations are possible. How those two postulates fit together has different interpretations, and is still up to debate for physicists.[22]

If the measurement base is not orthogonal, the measurements have unpleasant properties. It can happen that repeated measurements give different results, and the state keeps changing with the result every time. Orthogonal measurements ensure that the result

[20] Measurements with more physical background in [15, pp. 23-28].
[21] Measurements in different basses in [16, p. 22].
[22] Details about measurements given in [12, pp. 48-50].

will stay the same for repeated measurements. This is also one of the reasons that the computational base, $0, 1, 2, \ldots 2^q - 1$, is basically always used.[23]

It's always possible to perform only a single measurement of all qubits. This can be done by moving all measurements in a circuit to the end, and performing controlled versions of gates if they depend on outputs. Unneeded results can just be ignored. Measurements in different bases can also be combined, making only one collective measurement at the end sufficient.[24]

1.4 Oracles

To define problems that have a known advantage for quantum computers, often an oracle is used. A quantum oracle takes two registers as input. One is for holding the input of the query, and the other to hold the output of the oracle. The oracle evaluates some function f that will often have some special conditions to it. Since this has to be a reversible transformation, often $|x\rangle |0\rangle$ is given as input, and $|x\rangle |f(x)\rangle$ returned as output. Such an oracle can be represented as a box in the circuit model. The total transformation will often be called U_f. U_f can be treated like a normal gate in the circuit model.[25]

Oracles can be seen as a subroutine, where the internal workings are unknown or not specified. For this reason, they are sometimes referred to as a Blackbox. For the run-time, often only the number of queries to the oracle are counted. In many problems, access to an oracle is given, and the goal is to figure out some property of the function it evaluates with the least amount of queries possible.[26]

1.4.1 Simon's problem

As an example, Simon's problem will be presented. The oracle holds a function $f\colon \{0,1\}^n \to \{0,1\}^n$, and it's promised that

$$\forall x, y \in \{0,1\}^n : f(x) = f(y) \iff x \oplus y \in \{0, a\}$$

for some $a \in \{0,1\}^n$, where $\oplus$ is the bit-wise addition modulo 2. The problem is to find a.[27]

A classical computer can only search for a collision $x, y\colon f(x) = f(y)$. Since up to $2^{n-1} + 1$ values have to be queried, the run-time is $\Omega\left(2^{\frac{n}{2}}\right)$ in the worst case. A quantum computer, on the other hand, can perform the circuit shown in Figure 1.8. After the circuit, only elements orthogonal to a will be left in the state, and can be measured, as will be shown in Chapter 2. The circuit is repeated, each time learning an element from the set orthogonal to a. This is done until $n - 1$ linear independent elements are collected. With this, a can be calculated classically. One more test is needed to include

[23] Can be seen from general measurements of mixed states in [16, pp. 84-87].
[24] Combining measurements in [16, p. 86].
[25] Shown with the Deutsch algorithm as an example in [12, pp. 94-96].
[26] Complexity of oracle problems discussed in [5, p. 145].
[27] Simon's problem presented in [5, pp. 157-159].

the special case $a = 0$. All elements have an equal chance to be measured. The chance to get a new independent element gets smaller, the more elements are already known. For the last element to be measured, half of the possible elements are linear dependent and half are not. Therefore, the chance to get another one is $\frac{1}{2}$ in the worst case. This gives an average run-time of $O(n)$, an exponential improvement over the classical case.[28]

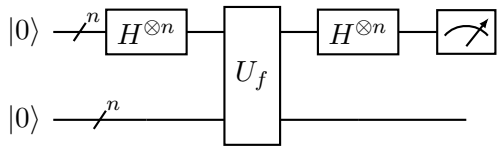


Figure 1.8: Circuit for solving Simon's problem.

The promise about the function the oracle evaluates is very important. Such a huge advantage for the quantum case is impossible without it. If the oracle could evaluate any function, and the quantum computer can solve the problem in time T, then a classical computer can do it in at most time T^6.[29]

1.5 Parallel Quantum Computers

In Figure 1.8, the first step of the algorithm is a layer of Hadamards. All the involved gates act on different qubits. The question arises if all of them could be executed in parallel. The answer is not a simple yes or no. This can be compared to a classical setting. A similar circuit with a layer of *NOT*, instead of H, could be constructed and the same question asked. This setting has the same answer. It can, if the hardware allows it. For a quantum computer, this is often assumed to be the case. Such a device will be called a parallel quantum computer. It is able to perform disjoint gates in parallel. The gates in a circuit can be bundled into layers that can be executed in parallel.[30]

For a fairer comparison of run-times, a classical version is used, the parallel computer. In this setting, larger inputs are not only granted more memory, but also more computation units, polynomial with the size of the input. Run-times can be much smaller on such a device, but could also not change at all, if the problem can't be parallelized.

This kind of device can be thought of as a supercomputer. A huge amount of cores is present, but access is only given to a fraction. How much depends on the problem size and importance.

[28] Solution explained in [12, pp. 103-107].
[29] This result and stronger ones for special cases in [4].
[30] Computation model and possible assumptions about actual devices discussed in [18].

1.6 Quantum Fourier Transform

Let $\omega_N := e^{\frac{i2\pi}{N}}$ be the N^{th} unit root. The classical discrete Fourier transform takes values $x_0, x_1, \ldots x_{N-1}$ and returns $y_0, y_1, \ldots y_{N-1}$ for

$$y_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n \omega_N^{-kn}$$

The quantum Fourier transform is very similar, but operates over a superposition. It takes $|x\rangle = \sum_{k=0}^{N-1} x_k |k\rangle$ and returns $\sum_{k=0}^{N-1} y_k |k\rangle$ where

$$y_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n \omega_N^{kn}$$

This operation can be performed very efficiently, compared to a classical fourier transform. However, while all the information is present to some degree, it can't be simply extracted. The y_k can not be measured. To get use out of this state, clever manipulations have to be performed, such that destructive interference occurs, and only desired results remain. The values can only be used relative to each other.[31]

The Hadamard gate is a kind of Fourier transform for a field with characteristic two. This explains why it turns up in so many solutions. To see this, let's apply a Hadamard to every qubit in a register $|x\rangle$

$$H^{\otimes q} |x\rangle = \sum_{n=0}^{2^q-1} x_n H^{\otimes q} |n\rangle = \sum_{n=0}^{2^q-1} x_n \frac{1}{\sqrt{2^q}} \sum_{k=0}^{2^q-1} (-1)^{k \cdot n} |k\rangle = \sum_{k=0}^{2^q-1} \frac{1}{\sqrt{2^q}} \sum_{n=0}^{2^q-1} x_n (-1)^{k \cdot n} |k\rangle$$

comparing this to the original Fourier transform with $N = 2^q$

$$\frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n (-1)^{k \cdot n} \qquad \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n \omega_N^{kn}$$

it's easy to see that they are the same formula, except for a different unit root.[32]

Using this kind of generalized Fourier transform, it is possible to solve the very general hidden subgroup problem. Given a group G, a subgroup $H \leq G$ and a set X. Let $f \colon G \to H$ with $\forall g_1, g_2 \in G \colon f(g_1) = f(g_2) \iff g_1 H = g_2 H$. It is promised that the function is constant on each coset of H, but different for all cosets of H. Assuming that G is known and oracle access to f is given, the problem is to find a generator set of H. A solution is known that uses only a polynomial amount of queries to the oracle. The solution is quit similar to the one for Simon's problem. Since Simon's problem is a very special case of this problem, this is not a big surprise. To solve the general problem, a Fourier transform is performed once before and after the oracle call. In general, this can take up to exponential time to execute. The Fourier transformation does not have to be easy to perform and can take this long.[33]

[31] Quantum Fourier transform explained with construction in [15, pp. 71-76].
[32] This relation details and more in [13].
[33] General hidden subgroup problem introduced in [16, pp. 240-242].

1.7 Other Topics

Some points that should not be unmentioned will now be presented.

It's not possible to copy a qubit. This is called the no-cloning theorem. Given two qubits, one that's supposed to be copied and a second it's supposed to be copied to $|\psi\rangle|0\rangle$, then there is no transformation with the result $|\psi\rangle|\psi\rangle$. However, there is a different kind of copy that is possible. If $|\psi\rangle = \sum_{k=0}^{2^q-1} \alpha_k |k\rangle$, then the state $\sum_{k=0}^{2^q-1} \alpha_k |k\rangle|k\rangle$ can be constructed using a *CNOT* with the control on ψ and the target on the register where the state should be copied to.[34]

There is a different way to represent transformations called density matrix. It allows the inclusion of uncertainty about what actual transformation is performed. This also works for the quantum states, allowing mixed states. They are not presented in detail, because only pure states are considered in this paper. They have the special form $|\psi\rangle\langle\psi|$ as a density matrix.[35]

Unstructured search is the problem of finding entries with some property in a search space. A classical computer would need time $O(N)$ for N entries. A quantum computer can do it in $O\left(\sqrt{N}\right)$ using Grover search. This is proven to be optimal.[36]

A nice representation for a single qubit is the Bloch-sphere. Nearly all states have a unique representation on it, and one-qubit gates can be represented by turns and flips of the sphere. There is however no known generalization to more than one qubit.[37]

It's possible to introduce randomness into oracles. The oracle will draw a random seed for every query, and evaluates a function dependent on the drawn seed. If this is done with Simon's problem, where the oracle evaluates one of all possible functions with the same a, the advantage for the quantum computer is even greater. The classical computer can't gain information from the oracle, since every result is purely random, while the quantum computer sees no difference. It is even possible to quantify when queries to an oracle are useless for a quantum computer in a classical setting.[38]

Different realization of qubits may be better suited for some applications than for others. Some could be better for performing calculations, while others may be better for transmitting or even storing data. This depends very much on their physical properties. It's not at all clear what qubits will look like in future systems.

[34] No-cloning explained in [12, pp. 216-217].
[35] Density operator introduced in [16, pp. 98-105].
[36] Grover search constructed in [16, pp. 265-271].
[37] Bloch sphere presented in [12, pp. 42-43].
[38] Oracle model with internal randomness and result from [11].

2 Generalized Simon's Problem

It's possible to allow more functions in Simon's problem, while keeping the run-time and even the solution for the quantum part the same. The promise to the function $f\colon \{0,1\}^n \to \{0,1\}^n$ can be relaxed. It only has to be promised that there exists a set $H \subseteq \{0,1\}^n$ such that

$$\forall x, y \in \{0,1\}^n : f(x) = f(y) \iff x \oplus y \in \langle H \rangle$$

where $\langle H \rangle$ is the span of H. The problem is now to find a base of $\langle H \rangle$. The original problem is the special case of $\langle H \rangle$ having at most dimension 1. This is still a special case of the hidden subgroup problem, but has a concrete and efficient solution.

This general version can be solved very similar to the original one. The quantum part will even stay exactly the same. The state before the measurements turns out to be an equal superposition over all elements orthogonal to all elements in the set $\langle H \rangle$, the elements in $\langle H \rangle^{\perp}$. The classical part has to be modified to deal with fewer dimensions in the measured subspace. Since the quantum circuit stays the same, the analysis for the special case can be generalized.[1]

WLOG $H = \langle H \rangle$ with dimension p, and let $K \subseteq \{0,1\}^n$ with $K \oplus H = \{0,1\}^n$ the direct sum. Such a K exists, and can be constructed by taking a set of representatives of $\{0,1\}^n_{/H}$.

The same circuit as for Simon's problem, shown in Figure 1.8, will be applied.

$$|0\rangle|0\rangle \overset{H^{\otimes n}\otimes I^{\otimes n}}{\to} \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle|0\rangle \overset{U_f}{\to} \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle|f(x)\rangle$$

$$\overset{H^{\otimes n}\otimes I^{\otimes n}}{\to} \frac{1}{2^n} \sum_{x\in\{0,1\}^n} \sum_{y\in\{0,1\}^n} (-1)^{x\cdot y} |y\rangle|f(x)\rangle = \sum_{y\in\{0,1\}^n} |y\rangle \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{x\cdot y} |f(x)\rangle$$

The likelihood of measuring result y in the first register is the sum of possibilities over all possible results with y in the first register. This is precisely the squared norm of the state with y fixed.

$$\left\| \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{x\cdot y} |f(x)\rangle \right\|^2 = \frac{1}{2^{2n}} \left\| \sum_{h\in H} \sum_{k\in K} (-1)^{(h\oplus k)\cdot y} |f(h\oplus k)\rangle \right\|^2$$

$$= \frac{1}{2^{2n}} \left\| \sum_{h\in H} (-1)^{h\cdot y} \sum_{k\in K} (-1)^{k\cdot y} |f(k)\rangle \right\|^2 = \frac{1}{2^{2n}} \left| \sum_{h\in H} (-1)^{h\cdot y} \right|^2 \left\| \sum_{k\in K} (-1)^{k\cdot y} |f(k)\rangle \right\|^2$$

[1] Analysis for the original case as in [5, pp. 157-164].

The sum over K is independent of y, since the $|f(k)\rangle$ are by definition all different to each other. The norm squared has the value 2^n, since $|f(k)\rangle$ has that many entries. This leaves only the sum over H to be evaluated. It holds that

$$\sum_{h\in H}(-1)^{h\cdot y}=\begin{cases}2^p & y\perp H\\ 0 & \text{else}\end{cases}$$

Prove by induction over p. The base case $p=0$ forces $H=\{0\}$.

$$\sum_{h\in H}(-1)^{h\cdot y}=\sum_{h\in\{0\}}(-1)^{h\cdot y}=1=\begin{cases}2^p & y\perp H\\ 0 & \text{else}\end{cases}$$

Given the assumption to hold for $p-1$, $p>0$ and $H=\langle h_1,\dots h_p\rangle$. The sum can be split along a base vector

$$\sum_{h\in H}(-1)^{h\cdot y}=\sum_{h\in\langle h_1,\dots,h_p\rangle}(-1)^{h\cdot y}=\sum_{h\in\langle h_1,\dots h_{p-1}\rangle}(-1)^{h\cdot y}+(-1)^{h_p\cdot y}\sum_{h\in\langle h_1,\dots h_{p-1}\rangle}(-1)^{h\cdot y}$$

$$=\begin{cases}2^{p-1}+(-1)^{h_p\cdot y}\,2^{p-1} & y\perp\langle h_1,\dots h_{p-1}\rangle\\ 0 & \text{else}\end{cases}=\begin{cases}2^p & y\perp\langle h_1,\dots h_p\rangle\\ 0 & \text{else}\end{cases}$$

Measuring after this procedure gives one of the elements in $\langle H\rangle^\perp$ with equal probability. Repeating this procedure often enough will result in a linear independent set with $n-p$ elements. This allows one to calculate a base of $\langle H\rangle$ classically.

The more linear independent elements are already known, the less likely it is to measure another one. For the last needed element, half the possible ones are dependent and half independent. Meaning, even in the worst case, the probability is $\frac{1}{2}$.

It's still possible to know for sure when a full base was found. The calculated base for H can be verified. Since the found set can only be missing elements, the calculated base for H can only be too large. By checking the function-values of all the calculated elements to be the same as $f(0)$, this can easily be tested.

Measuring about $100n$ elements, gives a probability of about $1-\frac{1}{2^{100}}$ to have found a full set. As a result, the average run-time is still in $O(n)$.

While not often mentioned, it turns out a generalized Simon's problem was already known. It was therefore, sadly, only rediscovered.[2]

[2]Found in [12, pp. 108-109].

3 Whitebox Problems

In the Bernstein-Vazirani problem, an oracle holds a linear function

$$f\colon \{0,1\}^n \to \{0,1\}, x \mapsto s \cdot x$$

for $s \in \{0,1\}^n$. The goal is to learn s. A classical computer needs at least n queries to the oracle, while a quantum computer only needs one.[1]

A non-oracle version of this problem was defined, the "2D Hidden Linear Function problem". A symmetric matrix A is given as input, and the problem is to find a z with $2z \cdot x \equiv x^T Ax \mod 4$, but only for x from a subset of all inputs. The defined problem is a restriction, where the possible inputs also fulfill a special condition. This restrictions makes the problem "2D". The problem turns out to be solvable with a constant depth quantum circuit, a shallow circuit. On the other hand, every classical circuit with bounded fan-in that solves the problem with a high probability has at least logarithmic depth.[2]

This result of superiority for quantum computation was improved multiple times.

At first, another problem was defined, allowing the worst-case superiority to be improved to average-case. Any circuit, with bounded fan-in that solves the new problem on a non-negligible fraction of the inputs, was shown to have logarithmic depth.[3]

Next, the run-time separation was further strengthened with two results. First, the restriction on the inputs can be relaxed to a kind of 1D version, while keeping the run-time. Second, even in the presence of noise, the advantage persists.[4]

It turns out that not even classical circuits with unbound fan-in can solve the problem with sub-logarithmic depth.[5]

The Bernstein-Vazirani problem is a Blackbox problem, since the inner workings of the oracle are unknown and even parts of it supposed to be deduced. For the 2D Hidden Linear function, there is no oracle and therefore no Blackbox. There is the input instead, a Whitebox, since the inner workings are fully known. The linear function is hidden in plain site.

The name "Whitebox" is inspired from software testing. Tests can be designed purely from a standpoint of supposed functionality, ignoring the inner workings of the function, called Blackbox-testing. If the implementation is given, the program flow can be considered too, allowing Whitebox-testing.

[1] Bernstein-Vazirani problem with solution given in [15, pp. 50-54].
[2] Definition of the problem and prove in [6].
[3] Average-case improvement from [9].
[4] Fewer restriction and noise from [7].
[5] Unbound fan-in and further results in [17].

3.1 Linear Function in Tensor

In the 2D hidden linear function problem, the Blackbox is replaced by an input in form of a matrix. To generalize this input, a tensor is used. For a tensor with rank k and n dimensions, the input will have n^k elements. The evaluated function was a kind of quadratic monomial. To generalize this, the function

$$f_A^c(x) := A \otimes_c x$$

where

$$A \otimes_{c+1} x := (A \otimes_c x) \otimes x$$

and

$$A \otimes_1 x := A \otimes x$$

will be used. Here $\otimes$ is the inner product on the last dimension. This is only sensible for $0 < c \leq k$. For $0 < c < k$, the function returns tensors with a rank less than k.

A is called symmetric, if and only if the order of indexes doesn't matter.

$$A_{i_1,\dots i_k} = A_{i_{\sigma(1)},\dots i_{\sigma(k)}}$$

for every permutation σ.

With enough restrictions, a linear function will be forced to be hidden in the tensor. This will allow white-boxing of some Blackbox-problems. As a first restriction, the tensors are assumed to be symmetric.

Using the symmetry of A and the binomial theorem, the function can be written as a sum with a linear part, for a $0 < c \leq k$.

$$f_A^c(x+y) = \sum_{i_1,\dots i_c} (x_{i_1} + y_{i_1}) \dots (x_{i_c} + y_{i_c}) A_{i_c \dots i_1}$$

$$= \sum_{0 \leq l \leq c} \begin{pmatrix} c \\ l \end{pmatrix} \sum_{i_1,\dots i_l} x_{i_1} \dots x_{i_l} \sum_{i_{l+1},\dots i_c} y_{i_{l+1}} \dots y_{i_c} A_{i_c \dots i_1}$$

If all but the first and last term are 0, then f is linear. Let

$$g := \gcd_{0<l<c} \begin{pmatrix} c \\ l \end{pmatrix}$$

the greatest common divisor over all the middle terms. Only the linear part remains modulo g.

$$f(x+y) \equiv f(x) + f(y) \mod g$$

If $g = 1$, the problem would turn trivial. The interesting cases are easily found using the closed form of g.[6]

$$g = \gcd_{0<l<c} \begin{pmatrix} c \\ l \end{pmatrix} = \begin{cases} p & c = p^b \text{ for } p \in \mathbb{P} \wedge b \in \mathbb{N} \\ 1 & \text{else} \end{cases}$$

[6] Shown and proven in [14].

Showing that precisely the prime powers are the interesting cases.

To be a linear function, it also has to hold that for all inputs x and possible a

$$f(ax) \equiv af(x) \mod g$$

$$\iff a^c f(x) \equiv af(x) \mod g$$

f has to be a multiple of g, or $a^c \equiv a \mod g$ at all times for this to hold. Since g is a prime and c a power of said prime, the second possibility is always the case. This is easily proven using Fermat's little theorem, $a^p \equiv a \mod p$.[7]

$$a^c \equiv a^{p^b} \equiv (a^p)^{p^{b-1}} \equiv a^{p^{b-1}} \equiv \ldots \equiv a^p \equiv a \mod p$$

No new condition is needed.

If c is chosen to be a prime power, the function is linear modulo that prime. f_A^c therefore hides a linear function of the form $\mathbb{F}_p^n \to \mathbb{F}_p^{n^{k-c}}$.[8] This means $z \in \mathbb{F}_p^{n^{k-c+1}}$ exists with $f_A^c(x) \equiv z \otimes_1 x \mod p$. For the special case $k = c$, this becomes $f_A^c(x) \equiv z \cdot x \mod p$.

Different restrictions to the inputs of the hidden function, and the tensors that are allowed as inputs, are possible and give different Whitebox-problems.

3.2 Simon's Whitebox

An even more general version of Simon's problem will be considered. For a prime p, the function will be of the form $f\colon \mathbb{F}_p^n \to \mathbb{F}_p^n$. The function and its inputs will therefore be evaluated modulo p. With some conditions, it will be possible to use $f(x) := f_A^{k-1}(x) = A \otimes_{k-1} x$ as such a function for a tensor A with rank k. The promise that has to be satisfied for all $x, y \in \mathbb{Z}_p^n$ is

$$f(x) = f(y) \iff x - y \in H$$

for some linear subspace H. It will always hold that $f(0) = 0$. The promise has to especially hold for $y = 0$.

$$f(x) = 0 \iff x \in H$$

Using this, the promise can be reformulated.

$$f(x) = f(y) \iff f(x - y) = 0$$

This can be combined into a single equation.

$$f(x) - f(y) = f(x - y)$$

Let's assume A to be symmetric, and use the result from Section §3.1.

[7] Fermat's little theorem proven in [10].

[8] Here $\mathbb{F}_p$ is the finite field with p elements.

$$f(x-y) \equiv f_A^{k-1}(x-y) \equiv f_A^{k-1}(x) - f_A^{k-1}(y) \equiv f(x) - f(y) \mod g$$

for

$$g := \gcd_{0<l<k-1} \left(\begin{array}{c} k-1 \\ l \end{array} \right)$$

If the rank of the tensor is chosen to be one more than a power of the prime p, $k = p^b + 1$, then the input can be used as an instance for this generalized version of Simon's problem. Simon's problem now boils down to finding the kernel of f.

The solution of the Blackbox version only works for $p = 2$. To use that solution, the oracle would have to be simulated. It isn't known if it's possible to do this efficiently.

3.3 Bernstein-Vazirani

Two ways will be proposed to realize a Whitebox-version of this problem. Both problems are relatively similar. The searched function is only linear for a subset of the possible inputs. This subset is chosen differently in the two versions.[9]

3.3.1 Kernel Version

Start with the function $f(x) := f_A^k(x) \mod 4$, where A is a symmetric tensor with rank k, a power of two. Consider the set

$$L_A := \left\{ x | A \otimes_{\frac{k}{2}} x = 0 \mod 2 \right\}$$

The set of inputs that are already 0 halfway through the calculation. It's a super-set of the actual kernel. This will be the set of inputs for the hidden linear function.

L_A is a linear subgroup.

$$\forall x, x' \in L_A \colon A \otimes_{\frac{k}{2}} \left(x + x'\right) \equiv A \otimes_{\frac{k}{2}} x + A \otimes_{\frac{k}{2}} x' \equiv 0 \mod 2$$

Using that $f_A^{\frac{k}{2}}$ is linear modulo two, as shown in Section §3.1.

$$\forall x, y \in L_A \colon f(x+y) = \sum_{0 \le c \le k} \left(\begin{array}{c} k \\ c \end{array} \right) \sum_{i_1, \ldots i_c} x_{i_1} \ldots x_{i_c} \sum_{i_{c+1}, \ldots i_k} y_{i_{c+1}} \ldots y_{i_k} A_{i_k \ldots i_1}$$

Using the symmetry of A. Because the product is applied $\frac{k}{2}$ times in the definition of L_A, it follows that

$$\sum_{i_1, \ldots i_c} x_{i_1} \ldots x_{i_c} A_{i_c \ldots i_1} \equiv 0 \mod 2$$

or

$$\sum_{i_{c+1}, \ldots i_k} y_{i_{c+1}} \ldots y_{i_k} A_{i_k \ldots i_{x+1}} \equiv 0 \mod 2$$

[9] Especially the second one is inspired by the hidden linear function problem from [6].

since one of the inputs is multiplied $\frac{k}{2}$ times. By factoring out another two from all the $\binom{k}{c}$, same as in Section §3.1, it's possible to deduce

$$\forall x, y \in L_A \colon f(x+y) \equiv f(x) + f(y) \mod 4$$

To be linear, it also has to hold that

$$\forall x \in L_A \colon f(ax) = af(x) \mod 4$$

This is simply true, because the inputs are from $\mathbb{F}_2^n$. Meaning, only $a = 0$ and $a = 1$ have to be considered.

Since f is linear modulo four, but constant zero modulo two for inputs from L_A, there exists $z \in \{0,1\}^n$ with $f(x) \equiv 2z \cdot x \mod 4$ for all $x \in L_A$. The problem is now to find such a z. This is the hidden linear function inside A. z is not unique, and the number of possible z depends on the dimension of L_A.

3.3.2 Linear Subgroup Version

Starting again with the function $f(x) = f_A^k \mod 4$, where A is a symmetric tensor with rank k, a power of 2. This time, consider the set

$$L_A := \{x \in \mathbb{F}_2^n | \forall y \in \mathbb{F}_2^n : f(x+y) \equiv f(x) + f(y) \mod 4\}$$

and let $K_A := L_A^\perp$. L_A is a linear subgroup. For all $x, x' \in L_A$ and $y \in \mathbb{F}_2^n$

$$f\left(x + x' + y\right) \equiv f(x) + f\left(x' + y\right) \equiv f(x) + f\left(x'\right) + f(y) \equiv f\left(x + x'\right) + f(y) \mod 4$$

The linear function basically exists by definition of L_A. Still

$$f(ax) = af(x) \mod 4$$

has to hold. But again, this is simply true because only $a = 0$ and $a = 1$ have to be considered.

There is again $z \in \{0,1\}^n$ with $f(x) \equiv 2z \cdot x \mod 4$ for all $x \in L_A$. z is not unique, and the dimension of the set of possible z is the same as the dimension of K_A.

It will be important that $K_A \oplus L_A = \mathbb{F}_2^n$. K_A is also a linear subspace, since

$$\forall x, x' \in K_A \forall y \in L_A \colon \left(x + x'\right) \cdot y = x \cdot y + x' \cdot y = 0$$

Given an element from $\mathbb{F}_2^n$, the parts from L_A and K_A can be uniquely separated using the Gram-Schmidt process. Therefore, $K_A \oplus L_A$ is the direct sum of $\mathbb{F}_2^n$.

3.3.3 Local Restriction

A restriction will be defined. It's possible to apply it to both versions. The inputs will be assumed to be from $\{0,1\}^{n^k}$, making the entries of the inputs bits. Instead of all symmetric tensors, only the ones that are "diagonally enough" are allowed. For A to be allowed as input, all entries of A are forced to be 0, except for some special ones. Only entries where the indexes have pairwise differences less than some constant m

$$\max_{1\leq a,b\leq k} \{|i_a - i_b|\} < m$$

can be 1. m can be thought of as 3, but will be written as a variable.

The problem still has the rank k and dimension n as variables. To simplify run-time analysis, k will also be assumed to be constant, but still written as a variable.

Technically, only the number of times a number appears as index of an entry not zero has to be capped. This special case is chosen, because it is quit easy to work with and visualize. The condition forces indexes to be close to each other, and therefore limits for how many elements an index can appear. In Figure 3.1, for the index j, all windows of size m, containing j, are depicted. Every possible index combination that contains

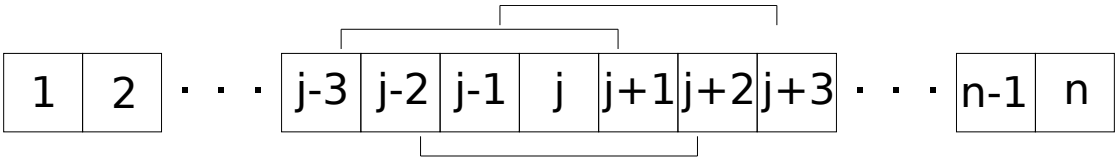


Figure 3.1: Index-windows for $m = 3$.

j has to be in one of the windows. There are m windows, and every window contains m^k index combinations, where some even don't contain j. Meaning at most m^{k+1} index combinations involving a specific index are possible.

One of the restricted problems turns out to be very easy to solve for a quantum computer.

3.4 Example

The example turns out to be for both versions, because L_A is the same set for both definitions. Let's consider the rank two tensor A.

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

First, calculate the set L_A. Meaning the set of all x with

$$Ax \equiv 0 \mod 2$$
$$\iff \begin{pmatrix} x_1 \\ 0 \end{pmatrix} \equiv 0 \mod 2$$

for the first version and

$$\begin{aligned} &\forall y \in \{0,1\}^2 : \ (x+y)^T A\,(x+y) \equiv x^T Ax + y^T Ay \mod 4 \\ \iff &\forall y \in \{0,1\}^2 : x^T Ay + y^T Ax \equiv 0 \mod 4 \\ \iff &\forall y \in \{0,1\}^2 : x_1 y_1 + y_1 x_1 \equiv 0 \mod 4 \end{aligned}$$

for the second. They give the exact same set L_A.

$$L_A = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}$$

While not needed, K_A can easily be calculated from L_A.

$$K_A = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right\}$$

The set of solutions is therefore given by

$$\begin{aligned} &\begin{pmatrix} 0 & x_2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ x_2 \end{pmatrix} \overset{!}{\equiv} 2 \begin{pmatrix} z_1 & z_2 \end{pmatrix} \begin{pmatrix} 0 \\ x_2 \end{pmatrix} \mod 4 \\ \iff &2 z_2 x_2 \equiv 0 \mod 4 \end{aligned}$$

giving

$$z \in \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right\}$$

as possible solutions.

The set of solutions has the same number of dimensions as K_A, and even is the same set for this example.

4 Solutions

The calculations are assumed to take place on a parallel quantum computer or classical parallel computer. In this setting, a bigger problem not only gets more memory, but also more calculation units, allowing it to perform more calculations in parallel, as introduced in Section §1.5. The local linear subgroup version will be solved.

4.1 Quantum Solution

The input is a symmetric tensor A with rank k, a power of two. k will be considered a constant and could be thought of as four. The linear function is hidden in

$$f(x) := \sum_{i_1,\dots,i_k} x_{i_1} \dots x_{i_k} A_{i_k \dots i_1} \mod 4$$

and takes inputs from $\mathbb{F}_2^n$. The inputs can therefore be considered as n bits.

The problem is similar enough to the "2D Hidden Linear Function problem" that the same solution can solve it, and the prove can be adapted.[1] The problem turns out to be a generalization, with a different restriction for locality. Instead of 2D, it is a 1D version, and instead of only symmetric matrices, symmetric tensors are used as inputs. This allows the proof to be generalized.

It's sufficient to calculate $i^{f(x)}$ to solve the problem. This can be done using multi controlled phase shifts

$$U_f := \prod_{1 \le i_1,\dots i_k \le n} C_{i_1,\dots i_k} S^{A_{i_1,\dots i_k}}$$

where $C_{i_1,\dots i_k} S^{A_{i_1,\dots i_k}}$ applies $C_{i_1,\dots i_k} S$ if $A_{i_1,\dots i_k}$ is one and otherwise the identity. $C_{i_1,\dots i_k} S$ in turn, applies a phase shift of i if all the $x_{i_1}, \dots x_{i_k}$ are one and otherwise the identity.

As a note, the indexes i_j are allowed to overlap. They can even be all the same. This makes the definition way simpler, but can be avoided. The product would have to be split into sub-products, each with a specific number of indexes involved. For completion, a closed form will be presented. For c the number of how many different indexes appear and $l_1, \dots l_n \in \mathbb{N}$ the number how often each index appears.

$$U_f := \prod_{1 \le c \le k} \prod_{\substack{l_1+\dots+l_n=k \\ \left|\left\{j | l_j > 0\right\}\right| = c}} \left(C_1^{l_1>0} \dots C_n^{l_n>0} S^{A \underbrace{1,1,\dots 1}_{l_1} \dots \underbrace{n,n,\dots n}_{l_n}} \right)^{\binom{k}{l_1,\dots l_n}}$$

[1] The mentioned problem, its solution and its proof in [6].

The exponent is the multinomial coefficient, and takes the multiplicity of the element into account. The control $C_j^{l_j>0}$ only acts if index j appears, meaning $l_j > 0$.

Given a state $|x\rangle = \sum_{l=0}^{2^n-1} \alpha_l |l\rangle$.

$$U_f |x\rangle = \sum_{l=0}^{2^n-1} \prod_{1\leq i_1,\dots i_k\leq n} C_{i_1,\dots i_k} S^{A_{i_1,\dots i_k}} \alpha_l |l\rangle = \sum_{l=0}^{2^n-1} \prod_{1\leq i_1,\dots i_k\leq n} i^{l_{i_1}\dots l_{i_k} A_{i_1,\dots i_k}} \alpha_l |l\rangle$$

$$= \sum_{l=0}^{2^n-1} i^{\sum_{i_1,\dots,i_k} l_{i_1}\dots l_{i_k} A_{i_k\dots i_1}} \alpha_l |l\rangle = \sum_{l=0}^{2^n-1} i^{f(l)} \alpha_l |l\rangle$$

U_f performs the desired transformation.

The considered version only overestimates how many controls are involved. It turns out that the amount of controls will dictate how many layers are needed, and therefore, how long the execution takes. The best run-time will only be overestimated.

Let $m \in \mathbb{N}$ be the constant from the locality condition of the problem. All the gates forming U_f would in general need up to k control bits, but at most m different indexes can be used. So at most m controls are actually needed in this sub-problem. Those operations have to be ordered in a way that needs as little layers as possible. One possibility is shown in Figure 4.1.

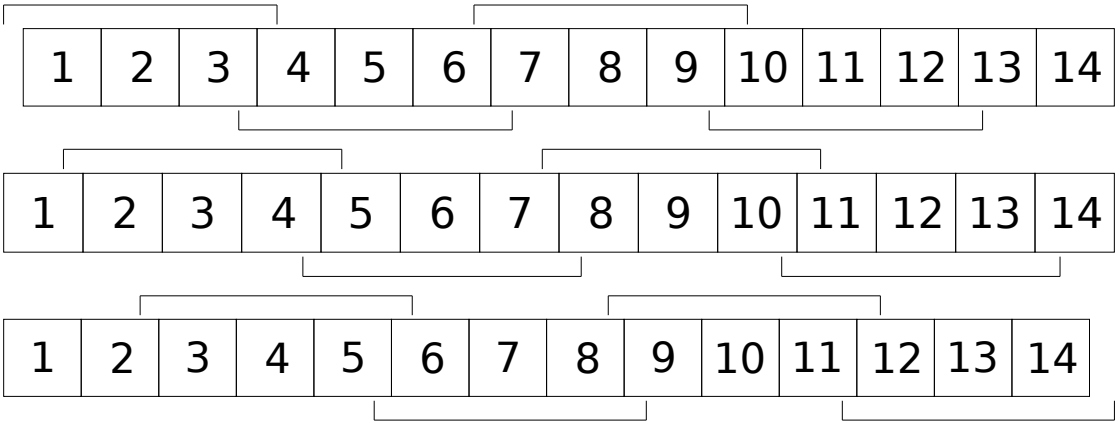


Figure 4.1: Possible order of windows for $m = 3$ and $n = 14$.

The indexes are grouped in windows of width m. Gates from different windows are disjoint, and can therefore be executed in parallel. The windows slide along the indexes, where $m - 1$ slides are needed to cover all possible combinations, making m times how many layers one window needs necessary. In every window, at most m^k combinations are possible, and therefore gates executed. Making at most m^{k+1} layers necessary to execute U_f.

Since the number of layers is capped by a constant, the parallel quantum computer can perform them in constant time. An algorithm with constant amount of layers is sometimes called a shallow circuit.

The full solution is $H^{\otimes n} U_f H^{\otimes n} |0\rangle$. This leaves the register in an equal superposition over all solutions. It will be shown, that only solutions can be measured from the produced state.

$$|0\rangle \overset{H^{\otimes n}}{\to} \frac{1}{\sqrt{2^n}} \sum_x^{2^n} |x\rangle \overset{U_f}{\to} \frac{1}{\sqrt{2^n}} \sum_x^{2^n} i^{f(x)} |x\rangle$$

$$\overset{H^{\otimes n}}{\to} \frac{1}{2^n} \sum_x^{2^n} i^{f(x)} \sum_y^{2^n} (-1)^{x \cdot y} |y\rangle = \frac{1}{2^n} \sum_y^{2^n} \sum_x^{2^n} i^{f(x)+2y \cdot x} |y\rangle$$

The probability to measure a z is therefore

$$\left| \frac{1}{2^n} \sum_x^{2^n} i^{f(x)+2z \cdot x} \right|^2 = \frac{1}{2^{2n}} \left| \sum_{x \in L_A} \sum_{y \in K_A} i^{f(x+y)+2z \cdot (x+y)} \right|^2$$

$$= \frac{1}{2^{2n}} \left| \sum_{x \in L_A} \sum_{y \in K_A} i^{f(x)+f(y)+2z \cdot x+2z \cdot y} \right|^2 = \left| \sum_{x \in L_A} i^{f(x)+2z \cdot x} \right|^2 \left| \sum_{y \in K_A} i^{f(y)+2z \cdot y} \right|^2$$

using that $\{0,1\}^n$ is the direct sum of L_A and K_A, and the property of elements in L_A.

For any solution of the problem y, the sum over L_A can be reshaped.

$$\sum_{x \in L_A} i^{f(x)+2z \cdot x} = \sum_{x \in L_A} i^{2y \cdot x+2z \cdot x} = \sum_{x \in L_A} (-1)^{(y \oplus z) \cdot x}$$

If now $y \oplus z \in K_A$

$$\sum_{x \in L_A} (-1)^{(y \oplus z) \cdot x} = \sum_{x \in L_A} 1 = |L_A|$$

this is the case if and only if z is a solution, since

$$\forall x \in L_A : f(x) \equiv 2y \cdot x \equiv 2y \cdot x + 2(y \oplus z) \cdot x \equiv 2z \cdot x \mod 4$$

Otherwise the sum can be split into parts from K_A and L_A, $y \oplus z = l \oplus k$ for $l \in L_A$ and $k \in K_A$.

$$\sum_{x \in L_A} (-1)^{(y \oplus z) \cdot x} = \sum_{x \in L_A} (-1)^{(l \oplus k) \cdot x} = \sum_{x \in L_A} (-1)^{l \cdot x + k \cdot x} = \sum_{x \in L_A} (-1)^{l \cdot x}$$

Since $l \oplus k \notin K_A = L_A^\perp$, there has to be an $l' \in L_A$ such that $(l \oplus k) \cdot l' = l \cdot l' = 1$. Let now $L_A = \langle l', l_1, \ldots l_p \rangle$ be a base of L_A.

$$\sum_{x \in L_A} (-1)^{l \cdot x} = \sum_{x \in \langle l', l_1, \ldots l_p \rangle} (-1)^{l \cdot x} = \sum_{x \in \langle l_1, \ldots l_p \rangle} (-1)^{l \cdot x} + (-1)^{l \cdot l'} \sum_{x \in \langle l_1, \ldots l_p \rangle} (-1)^{l \cdot x} = 0$$

This shows that only solutions have a non-zero amplitude, and can therefore be measured. They are also evenly distributed, because

$$\sum_{y \in K_A} i^{f(y)+2z \cdot y} = \sum_{y \in K_A} i^{4z \cdot y} = \sum_{y \in K_A} 1 = |K_A|$$

for a solution z. It's possible to see from this that the set of solutions and K_A have the same dimension.

4.2 Example Revisited

For

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

the possible solutions turned out to be

$$z \in \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right\}$$

The vectors are represented by $|x_2 x_1\rangle$. Now, the proposed solution is depicted in Figure 4.2.

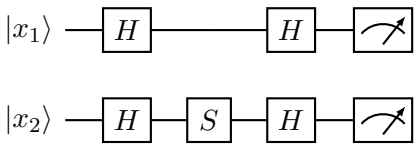


Figure 4.2: Example solution.

The, not normalized, state after each layer is

$$\begin{aligned} |00\rangle &\overset{H^{\otimes 2}}{\to} |00\rangle + |01\rangle + |10\rangle + |11\rangle \overset{I \otimes S}{\to} |00\rangle + i\,|01\rangle + |10\rangle + i\,|11\rangle \\ &\overset{H^{\otimes 2}}{\to} (|00\rangle + |01\rangle + |10\rangle + |11\rangle) + i\,(|00\rangle - |01\rangle + |10\rangle - |11\rangle) \\ &\quad + (|00\rangle + |01\rangle - |10\rangle - |11\rangle) + i\,(|00\rangle - |01\rangle - |10\rangle + |11\rangle) \\ &= (2 + 2i)\,|00\rangle + (2 - 2i)\,|01\rangle + 0\,|10\rangle + 0\,|11\rangle \\ &= (2 + 2i)\,|00\rangle + (2 - 2i)\,|10\rangle \end{aligned}$$

Therefore, measuring will give one of the solutions with equal probability.

4.3 Classical Solution

The best known, classical solution is the simulation of the quantum one. For $k = 4$ this works efficiently, but it's possible that harder cases exist. For the function

$$f(x) = \sum_{i_1,\ldots,i_k} x_{i_1} \ldots x_{i_k} A_{i_k \ldots i_1} \mod 4$$

the elements that are summed up a multiple of four times can be ignored. The special case of $k = 4$ will be considered.

$$f(x) = \sum_{a,b,c,d} x_a x_b x_c x_d A_{dcba} \mod 4$$

To know if an element has to be considered, the form of its index-set is important. Elements with the same indexes and multiplicity of indexes have to be equal, because of the symmetry of A. Meaning, what indexes and how often they appear is important, not their order. The possible multiplicities and how many entries are forced to be equal is shown in Table 4.1. This means, only the entries with all indexes equal and pairs of

Index combination	Number of same elements
1–1–1–1	4!
1–1–2	$4 \cdot 3$
1–3	4
2–2	$\frac{4 \cdot 3}{2} = 6$
4	1

Table 4.1: Possible multiplicities with their amount.

indexes have to be considered for the function.

The quantum algorithm can therefore be simplified.

$$U_f := \prod_{1 \leq i \leq n} S_i^{A_{i,i,i,i}} \prod_{1 \leq i < j \leq n} C_{i,j} Z^{A_{i,i,j,j}}$$

Here $C_{i,j}Z$ performs a Z on qubit j if qubit i is one. There is a special subgroup that contains these transformations, the Clifford-group. The group is special, because it's operations can be simulated efficiently. The gates *CNOT*, H and S generate all transformations in the Clifford-group.[2]

CZ can be constructed as follows:

$$(S \otimes S)\left(CNOT_{2,1}\left(I \otimes SSS\right) CNOT_{2,1}\right)$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & i & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -i & 0 & 0 \\ 0 & 0 & -i & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

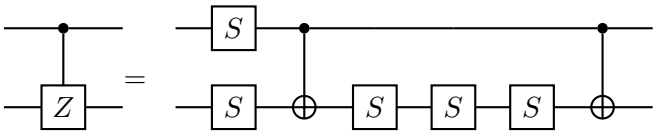


Figure 4.3: Realization of *CZ* in the Clifford-group.

Here $CNOT_{i,j}$ executes the *CNOT* with target i and control j.

The simulation has to perform up to n S, $n^2 - n$ *CZ* and $2n$ H gates. A total of $O\left(n^2\right)$ gates plus measurements is needed. All n qubits have to be measured at the end.

[2] Introduction to the Clifford-group in [2, pp. 1-2].

The idea for the simulation is as follows. A state can always be represented in turns of eigenvalue equations. This makes it possible to not represent a state as sum of base states, but in terms of stabilizers. In this special group, this turns out to be way more efficient. Only a polynomial amount of stabilizers is needed to uniquely define a state in this group, instead of an exponential amount of base states.[3]

Let's analyze the run-time of a specific simulation.[4]

The algorithm holds a $2n \times (2n+1)$ binary matrix to represent the state. All the gates perform independent operations on all rows. Therefore, it's possible to perform every gate in constant time on a parallel computer. They all write on the last entry, so every gate has to be performed individually. A total time of $O\left(n^2\right)$ will be needed for the gates.

Measurements are not as easily done in parallel. There are two possible types of measurement. The kind that changes the state and the type that doesn't.

The first reads potentially from the whole table, but only writes into one row. It also has to calculate two sums of n elements, meaning one measurements takes time $O\left(\ln\left(n\right)\right)$. Since the reads have to wait for the writes, the measurements can only be performed one at a time, taking up to $O\left(n\ln\left(n\right)\right)$ if all measurements are of this type.

The other type only writes into a new, temporary row, and can therefore be performed in parallel. However, one measurements has to calculate n sums of n elements, one after the other. Therefore, a run-time up to $O\left(n\ln\left(n\right)\right)$ is needed.

A total run-time of $O\left(n^2\right)$ is therefore possible on a parallel computer.

This only works since CZ, S, and H are in the Clifford-group. For bigger problems, gates like CS or CCZ could be needed. Both are not in the Clifford-group, since it would be universal with either of them.[5] Therefore, this sort of simulation is not possible anymore, and a different approach has to be used.

The existence of such a dimension depends on the general version of Table 4.1. U_f is not in the Clifford-group, if and only if an entry exists with one of two possible forms of indexes. For the number of different indexes c and the number of orders o:

1. $o \equiv 1 \mod 2$ and $c > 1$
2. $o \equiv 2 \mod 4$ and $c > 2$

If there is no entry with such an index, all of them either have $o \equiv 0 \mod 4$ and can be ignored, or only need S and CZ to be simulated. If such an entry does exist, it needs at least CCZ or CS to be simulated.

The entries can be described as follows. For any $l_1 + \cdots + l_n = k$

$$o = \begin{pmatrix} k \\ l_1, \ldots l_n \end{pmatrix}$$

and $|\{j|l_j > 0\}| = c$, are precisely the possibilities. This means c indexes are present and there are o entries corresponding to this index-set. The question can now be reformulated to the existence of an index-set with one of the needed criterion.

[3] Introduction for using stabilizers to represent the state in [2, pp. 2-3].
[4] Using the simulation from [2, p. 4].
[5] Mentioned and used for the proof in [3].

Bibliography

[1] Scott Aaronson. *Quantum Computing since Democritus*. Cambridge University Press, 2013. ISBN: 978-0-521-19956-8. URL: http://www.cambridge.org/de/academic/subjects/physics/quantum-physics-quantum-information-and-quantum-computation/quantum-computing-democritus?format=PB.

[2] Scott Aaronson and Daniel Gottesman. "Improved Simulation of Stabilizer Circuits". In: *Phys. Rev. A 70, 052328 (2004) (14 pages)* (June 25, 2004). DOI: 10.1103/PhysRevA.70.052328. arXiv: quant-ph/0406196v5 [quant-ph].

[3] Dorit Aharonov. *A Simple Proof that Toffoli and Hadamard are Quantum Universal*. Jan. 9, 2003. arXiv: quant-ph/0301040v1 [quant-ph].

[4] Robert Beals et al. *Quantum Lower Bounds by Polynomials*. Feb. 18, 1998. arXiv: quant-ph/9802049v3 [quant-ph].

[5] Chris Bernhardt. *Quantum Computing for Everyone (The MIT Press)*. The MIT Press, 2019. ISBN: 978-0262039253. URL: https://mitpress.mit.edu/books/quantum-computing-everyone.

[6] Sergey Bravyi, David Gosset, and Robert König. "Quantum advantage with shallow circuits". In: *Science* 362.6412 (2018), pp. 308–311. ISSN: 1095-9203. DOI: 10.1126/science.aar3106.

[7] Sergey Bravyi et al. *Quantum advantage with noisy shallow circuits in 3D*. Apr. 2, 2019. arXiv: 1904.01502v1 [quant-ph].

[8] Kai-Wen Cheng and Chien-Cheng Tseng. *Quantum Plain and Carry Look-Ahead Adders*. June 5, 2002. arXiv: quant-ph/0206028v1 [quant-ph].

[9] François Le Gall. *Average-Case Quantum Advantage with Shallow Circuits*. Oct. 30, 2018. arXiv: 1810.12792v4 [quant-ph].

[10] S. W. Golomb. "Combinatorial Proof of Fermat's "Little" Theorem". In: *The American Mathematical Monthly* 63.10 (1956), pp. 718–718. ISSN: 00029890, 19300972. URL: http://www.jstor.org/stable/2309563.

[11] Aram W. Harrow and David J. Rosenbaum. "Uselessness for an Oracle Model with Internal Randomness". In: *Q. Inf. Comput. vol. 14, no. 7&8, pp. 608-624 (2014)* (Nov. 7, 2011). arXiv: 1111.1462v2 [quant-ph]. URL: https://arxiv.org/abs/1111.1462.

[12] Phillip Kaye, Raymond Laflamme, and Michele Mosca. *An Introduction to Quantum Computing*. Oxford University Press, 2007. ISBN: 978-0-19-857000-4. URL: https://www.amazon.com/Introduction-Quantum-Computing-Phillip-Kaye/dp/0198570007?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0198570007.

[13] Kunz. "On the Equivalence Between One-Dimensional Discrete Walsh-Hadamard and Multidimensional Discrete Fourier Transforms". In: *IEEE Transactions on Computers* C-28.3 (Mar. 1979), pp. 267–268. ISSN: 2326-3814. DOI: 10.1109/TC.1979.1675334.

[14] Carl McTague. "On the Greatest Common Divisor of Binomial Coefficients n choose q, n choose 2q, n choose 3q, dots". In: *Amer. Math. Monthly, Vol. 124, No. 4 (April 2017), pp. 353-356* (Oct. 22, 2015). DOI: 10.4169/amer.math.monthly.124.4.353. arXiv: 1510.06696v5 [math.CO].

[15] N. David Mermin. *Quantum Computer Science: An Introduction*. Cambridge University Press, 2007. ISBN: 978-0-521-87658-2. URL: https://www.amazon.com/Quantum-Computer-Science-David-Mermin/dp/0521876583?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0521876583.

[16] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. 10$^{\text{th}}$. New York, NY, USA: Cambridge University Press, 2011. ISBN: 1107002176, 9781107002173.

[17] Adam Bene Watts et al. "Exponential separation between shallow quantum circuits and unbounded fan-in shallow classical circuits". In: *Proceedings of the 51st Annual Symposium on Theory of Computing, STOC 2019, Pages 515-526* (June 20, 2019). DOI: 10.1145/3313276.3316404. arXiv: 1906.08890v1 [quant-ph].

[18] Yu Zhang et al. "Optimizing Quantum Programs Against Decoherence: Delaying Qubits into Quantum Superposition". In: *2019 International Symposium on Theoretical Aspects of Software Engineering (TASE)* (July 2019). DOI: 10.1109/tase.2019.000-2.