SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Efficient Implementation of Constant Propagation for Quantum Circuits

Jakob Zuchna

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Efficient Implementation of Constant Propagation for Quantum Circuits

Effiziente Implementierung von Konstanten-Propagation für Quanten-Schaltkreise

Author:	Jakob Zuchna
Supervisor:	Prof. Dr. Helmut Seidl
Advisor:	M.Sc. Yannick Stade
Submission Date:	15.05.2023

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

Munich, 15.05.2023 Jakob Zuchna

Abstract

Quantum Computing is a technology that promises a significant performance increase for specific problems like prime factorization or machine learning. Due to real quantum computers still having significant error when executing operations, it is important to reduce quantum circuits to a minimum, before being run. Earlier approaches use quantum computers to optimize, take exponential time or only allow qubits to be in pure states to continue optimizing. We introduce an algorithm that uses bitwise simulation of the state of all qubits up to a certain complexity, to stay in polynomial time, in order to reduce the complexity of quantum circuits, removing superfluous control qubits or gates. This implementation is a version in `C++` of the same algorithm that will be proposed in a to be published paper implemented in `OCAML`, which is around 50% faster on median.

Contents

1. Introduction

Quantum computing is an emerging technology that promises significantly improved efficiency for a variety of applications. Prime factorization in polynomial time [13], improved efficiency in neural networks [2] or database search [6] just to name a few.

The current state of real world quantum computers is called the Noisy intermediate-scale quantum (NISQ) Era [11]. This means that while quantum computers may offer 50-100 qubits, the size of the circuits that can be accurately executed will be limited by the reliability of the results. This necessitates the reduction of complexity of circuits during the compilation.

One approach to do this is to use ZX-calculus [1], which is able to find a minimal circuit that is still equal to the original one, but it takes exponential time. If we know the state of a qubit at a specific point in the circuit, we can decide if a control qubit or gate is superfluous. This can for example be done using a quantum computer [8] and running the circuit up to a specific point. Another way to do this is to keep track of the state of a qubit [9] and finding gates that can be simplified.

We propose an algorithm that simulates the state of qubits using a hash map that stores non-zero amplitudes of the state similar to [5]. We only combine the states of qubits into a hash map if they have interacted with each other, putting qubits into entanglement groups. We limit the number of amplitudes in a state to a fixed number (N_{max}) to not simulate the entire circuit (exponential), but rather simulate up to a certain complexity and stay in polynomial time. The state is then used to remove implied and therefore superfluous control qubits as well as non-activating gates. Very small amplitudes that are below a certain threshold (ε) are dropped. We argue that very small amplitudes would not change the result of the circuit on a real quantum computer, as they are not distinguishable from noise.

This implementation is done in `C++`. This is due to the idea that it would be a more efficient version of another implementation in `OCAML` of a yet to be published paper by Chen and Stade [3]. We show that the version in `C++` is on median around 50% faster than the `OCAml` version.

In chapter 2 we describe the theoretical background as well as going into further detail about related works on this topic. The details of the implementation are described in chapter 3. In chapter 4 we evaluate the performance of the algorithm using the Munich Quantum Toolkit (MQT) benchmark [12], as well as showing the influence of

N_{max} and ε on the output, before drawing a conclusion in chapter 6.

2. Background

2.1. Quantum Computing

Quantum Computing uses qubits (quantum bits) as its unit of information, which is distinct from classical bits, as they can exist in a superposition of states represented as $|\psi\rangle = c_0 |0\rangle + c_1 |1\rangle$, where c_0 and c_1 are complex numbers. These states can be described as vectors with $|0\rangle = \binom{1}{0}$, $|1\rangle = \binom{0}{1}$ and a superposition $|\psi\rangle = c_0 |0\rangle + c_1 |1\rangle = \binom{c_0}{c_1}$ [7].

The outcome of a measurement on qubits is only a single state. No information about the amplitudes can be extracted in a single measurement. The probability of a state being measured is $|c|^2$. Therefore the sum of the probabilities is always 1:

$$\sum_i |c_i|^2 = 1$$

The state of multiple qubits can be described as the tensor product of their states like:

$$|\psi_{12}\rangle = |\psi_1\rangle \otimes |\psi_2\rangle = (\alpha_1 |0\rangle + \beta_1 |1\rangle) \otimes (\alpha_2 |0\rangle + \beta_2 |1\rangle) = \begin{pmatrix} \alpha_1\alpha_2 \\ \alpha_1\beta_1 \\ \beta_1\alpha_2 \\ \beta_1\beta_2 \end{pmatrix}$$

The size of the state vector for n qubits is 2^n. We can also describe this vector as a linear combination:

$$|\psi_{12}\rangle = \begin{pmatrix} \alpha_1\alpha_2 \\ \alpha_1\beta_1 \\ \beta_1\alpha_2 \\ \beta_1\beta_2 \end{pmatrix} = \alpha_1\alpha_2 |00\rangle + \alpha_1\beta_1 |01\rangle + \beta_1\alpha_2 |10\rangle + \beta_1\beta_2 |11\rangle = \sum_{n=0}^{2^2-1} c_n |n\rangle$$

To change the state of a qubit, we apply gates. Single qubit gates are described by 2x2 matrices. The simplest example for this is the identity operator I:

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

To put a qubit in a superposition the Hadamard Gate (H) can be used. It is defined by the matrix:

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

The state of a qubit after applying a gate is determined by matrix multiplication

$$H|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\cdot 1 + 0\cdot 1\\ 1\cdot 1 - 0\cdot 1\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ 1\end{pmatrix} = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right)$$

Another example is the X Gate (also denoted as $\oplus$). It acts similar to a logical *NOT*:

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

To apply a gate to only a single qubit in a qubit state with n qubits, we look at the state as a linear combination of vectors $|i\rangle$ each with an amplitude c_i. Let $i[k]$ be the k th bit of i and $i[k] := b$ denote i with the k th bit set to b. We can then select which column, of the matrix U that represents the gate to apply by the value of $i[t]$, where t is the index of the target qubit.

$$U|\psi\rangle = \begin{pmatrix}m_{00} & m_{01}\\ m_{10} & m_{11}\end{pmatrix}\sum_{i=0}^{2^n-1} c_i|i\rangle = \sum_{i=0}^{2^n-1}\begin{cases}m_{00}c_i|i\rangle + m_{10}c_i|i[t] := 1\rangle & i[t] = 0\\ m_{11}c_i|i\rangle + m_{01}c_i|i[t] := 0\rangle & i[t] = 1\end{cases}$$

To have multiple qubits interact, we can use controlled gates. If the control qubit is in state $|0\rangle$, the gate is not applied. If it is in state $|1\rangle$, the gate is applied. One example for a controlled gate is the *CX* (*CNOT*) gate. As the name suggests, it applies an X gate depending on the control qubit. Controlled gates can also be described as matrices:

$$CX = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X = \begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix}\otimes I + \begin{pmatrix}0 & 0\\ 0 & 1\end{pmatrix}\otimes X = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}$$

Controlled qubits have a control qubit, which determines if the gate is activated, and a target qubit, to which the gate is applied to if activated. Let us apply the *CX* gate to a qubit state:

$$CX|\psi\rangle = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}\begin{pmatrix}\alpha_{00}\\ \alpha_{01}\\ \alpha_{10}\\ \alpha_{11}\end{pmatrix} = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}\begin{pmatrix}\alpha_{00}\\ \alpha_{01}\\ \alpha_{11}\\ \alpha_{10}\end{pmatrix}$$

We see that for the amplitudes where the control qubit was $|1\rangle$, the target qubit was toggled, as the X gate was applied to it.

Let us apply this to a simple circuit:

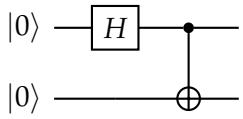

Figure 2.1.: Simple quantum circuit to entangle qubits

After applying the H Gate, we can describe the state as

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle$$

Applying the CX gate, we find the state of the two qubits to be

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

This state can no longer be described as a tensor product of single qubit states. When this is the case we call two qubits *entangled*.

We can also generalize the CX gate to a generic controlled gate CU, described by a 2x2 Matrix:

$$CU\,|\psi\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & m_{00} & m_{01} \\ 0 & 0 & m_{10} & m_{11} \end{pmatrix} \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix} = \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ m_{00}\alpha_{10} + m_{01}\alpha_{11} \\ m_{10}\alpha_{10} + m_{11}\alpha_{11} \end{pmatrix}$$

Again, we see that the generic gate U is only applied to the target qubit if the control qubit is in $|1\rangle$. Therefore we could also split this calculation into the "activated" part and the "not activated" part.

Gates can also have multiple control qubits. A gate only activates if *all* control qubits are $|1\rangle$. Let C be the set of control qubits. We first split the amplitudes into two groups, based on whether the gate will be activated. We then only apply the gate, where it should be activated and sum up the resulting amplitudes.

$$U(C)\,|\psi\rangle = (U\,|\psi\rangle\,, \forall c \in Ci[c] = 1) + (|\psi\rangle\,, \exists c \in Ci[c] = 0)$$

2.2. Related Work

Every gate executed on a quantum computer induces error. Current quantum computers have an error rate of about 0,66 % [15] for every execution of a CX gate. Therefore a lot of approaches have been proposed how to reduce quantum circuits while still preserving functionality. They are usually executed by a compiler.

2.2.1. Relaxed Peephole Optimization

Relaxed Peephole Optimization (RPO) [9] traverses through the quantum circuit and keeps track of the state of a single qubit. A single qubit can only be in one of 6 basis states. If a gate is applied that would put the qubit in a different state, it is assigned $\top$ (unknown). Then the single-qubit state is used to find specific patterns in the circuit that can be simplified or removed.

2.2.2. AQCEL

AQCEL (Advancin Quantum Circuits by ICEPP and LBNL) [8] determines the state of a qubit using a quantum computer. The circuit is run up to a specific point and then measured. After a number of runs, the likelihood of each result is determined. If a result is very unlikely it is omitted. The results are then used to determine if a controlled gate can be simplified or removed. They show that a certain range of the threshold for omitting results (in their case $\sim 0,15$) generates an optimized circuit, that when run on a quantum computer, produces measurements that are closer to the (error-free) simulation of the original circuit than the original circuit itself.

2.2.3. QSystem

QSystem [5] is a bitwise simulator for quantum circuits. It uses a hash map to represent the state of a qubit, where each entry corresponds to a non-zero amplitude. This means that the size does not exponentially increase with a growing number of qubits, but rather with the number of basis states needed to describe the system. QSystem supports not arbitrarily controlled gates, but only CX. They show that for specific circuits like GHZ (Entangled qubits with two basis states), the hash map structure vastly outperforms other structures like a vector or a matrix.

3. Implementation

3.1. Open Quantum Assembly Language (QASM)

As our input and output format we use Open QASM [4] (`.qasm`) files. We use the Quantum Functionality Representation (QFR) [10] as our parser. It is part of the Munich Quantum Toolkit (MQT). It currently supports OpenQASM 2.0.

3.2. Circuit Representation

The parsed circuit is stored as a `std::vector` of gates. Each gate has a type, target qubits and control qubits. The supported types are almost all types that are defined in the standard OpenQASM header file. Gates that are not supported or custom defined gates consisting of several other gates are flattened. This leaves us with a structure that we can easily iterate over. Unfortunately deleting in the middle of the vector is in $\mathcal{O}(n)$, where n is the number of elements after the to be deleted one [14]. This makes changing the vector unfeasible. Therefore we need to copy the gates to a new vector. A data structure like a linked list would be better for our application as we do not need random access and deletion would be in constant time, as well as making copying gates unnecessary.

To obtain the matrix of a gate, we implemented a function that takes the type and parameters of a gate and returns the corresponding matrix. The generic gate U for OpenQASM is defined as [4]:

$$U(\theta, \phi, \lambda) := \begin{pmatrix} e^{-i(\phi+\lambda)/2}\cos(\theta/2) & -e^{-i(\phi-\lambda)/2}\sin(\theta/2) \\ e^{i(\phi-\lambda)/2}\sin(\theta/2) & e^{i(\phi+\lambda)/2}\cos(\theta/2) \end{pmatrix} \tag{3.1}$$

Even though all gates could be expressed as a $U3$ gate, the more simple ones are just defined as fixed numbers. Similiar to the single-qubit gates, the matrices of the two-qubit gates are also defined in a function returning a 4x4 matrix. All exact definitions of the matrices are in tables A.1 and A.2 in the appendix.

3.3. Data Structure

A qubit state of n entangled qubits is stored in a hash map (`std::unordered_map`), where we only store basis states with non-zero amplitudes.

$$|\psi\rangle = \sum_{i=0}^{2^n-1} c_i |i\rangle = \sum_{i=0}^{2^n-1} qubit[i] |i\rangle$$

Where the key of the hash map is a binary number of arbitrary length (`std::vector<bool>`), and the value is a complex number (`std::complex<double>`). Both types are wrapped in custom classes to support implementation specific methods (e.g. `std::vector<bool>` does not support bit wise logic operations). Using a hash map, our memory usage is in $\mathcal{O}(2^n)$, with n being the number of qubits, but is now dependent on the number of non-zero basis states in the qubit state. If the complexity of the state is too high (too many possible basis states), we switch the state of the qubit to $\top$ (i.e. not known).

To make use of the fact that not all qubits might interact with each other, we split them into groups of qubits that have interacted and therefore might be entangled. We store the state of all qubits in an array, where each element holds either $\top$ or a pointer to the hash map of the corresponding entanglement group.

In order to illustrate this concept we evaluate a simple circuit which can be seen in figure 3.1. Note that measurements also cause qubits to be in the $\top$ state.

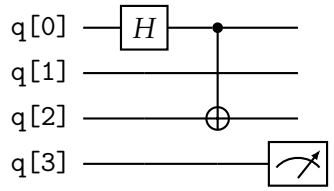

Figure 3.1.: Simple Example for a quantum circuit

We show the resulting state in figure 3.2. Qubits `q[0]` and `q[2]` have interacted via the cx gate and therefore point to the same hash map. No gates were applied to `q[1]`. It is therefore still in the initial $|0\rangle$ state, which is a hash map with only one entry.

3.4. Propagation

3.4.1. Single-Qubit Gates

To apply a single-qubit gate to a target qubit t, we select the column of the matrix based on what value the target qubit has in that state. $i[k]$ denotes the k th bit of i. We do this

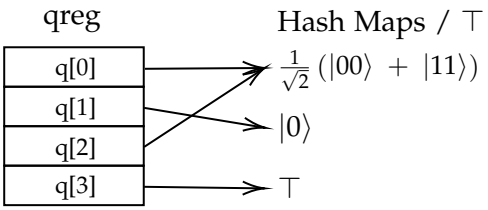

Figure 3.2.: State of Data Structure after simple example

with algorithm 1. For each amplitude α_b of a basis state $|b\rangle$ we apply the corresponding elements of the Matrix M that describes the gate. We thereby calculate the new state to be:

$$M|\psi\rangle = \begin{pmatrix} m_{00} & m_{01} \\ m_{10} & m_{11} \end{pmatrix} \sum_{b=0}^{2^n-1} \alpha_b |b\rangle = \sum_{b=0}^{2^n-1} \begin{cases} m_{00}\alpha_b |i\rangle + m_{10}\alpha_b |b[t] := 1\rangle & \text{if } b[t] = 0 \\ m_{11}\alpha_b |i\rangle + m_{01}\alpha_b |b[t] := 0\rangle & \text{if } b[t] = 1 \end{cases}$$

Algorithm 1 Apply a single qubit gate with matrix M to a qubit state $|\psi_O\rangle$ on qubit with index t

function ApplySingleQubitGate($|\psi_O\rangle, M, t$)
  $\begin{pmatrix} m_{00} & m_{01} \\ m_{10} & m_{11} \end{pmatrix} \leftarrow M$
  $|\psi\rangle \leftarrow \varnothing$ ▷ Initialize new state
  for $(b, \alpha) \in |\psi_O\rangle$ **do**
    if $b[t]$ **then**
      $|\psi\rangle[b] \leftarrow |\psi\rangle[b] + m_{11} \cdot \alpha$
      $b[t] \leftarrow 0$
      $|\psi\rangle[b] \leftarrow |\psi\rangle[b] + m_{01} \cdot \alpha$
    else
      $|\psi\rangle[b] \leftarrow |\psi\rangle[b] + m_{00} \cdot \alpha$
      $b[t] \leftarrow 1$
      $|\psi\rangle[b] \leftarrow |\psi\rangle[b] + m_{10} \cdot \alpha$
  return $|\psi\rangle$

3.4.2. Combine Qubit States

When multiple qubits interact, they might be entangled. We therefore combine the qubit states before applying the actual operation that uses both. We do this by calculating the

tensor product of both states. Qubit states only know the index of a qubit within the state. Therefore when combining two entanglement groups we need to find the order of the indices in the new state. This is done in the following steps:

1. Find indices of each entanglement group in the `qreg`
2. Find order in which the next higher bit of each state should be put in the new bit string. For example "010" - First index of q_0, First index of q_1, second index of q_0.
3. For each combination of basis states in the tensor product find the new bit string, multiply the amplitudes and add it to the new state

The actual implementation takes a lot of bitwise manipulation, therefore the algorithm is here only described using the rough steps in which it operates.

3.4.3. Two Qubit Gates

As for a single qubit gate, we can very similarly apply a two qubit gate. The only difference is that matrix M describing the operator is now a 4x4 matrix. Therefore we do not use `if/else` to determine the used matrix entry, but rather find the column by calculating its index from the bitstring and then going over all rows of the matrix.

Algorithm 2 Apply a two qubit gate with Matrix M to qubit state $|\psi_O\rangle$ with targets t_1 and t_2

function ApplyTwoQubitGate($|\psi_O\rangle, M, t_1, t_2$)
  $|\psi\rangle \leftarrow \varnothing$ ▷ Initialize new state
  for $(b, \alpha) \in |\psi_O\rangle$ **do**
    $col \leftarrow b[t_1] + b[t_2] << 1$ ▷ Find column index $\in [0..3]$
    for $row \in [0..3]$ **do**
      $n \leftarrow b$
      $n[t_1] = row[0]$
      $n[t_2] = row[1]$
      $|\psi\rangle\,[n] \leftarrow |\psi\rangle\,[n] + M[col][row] \cdot \alpha$
  return $|\psi\rangle$

3.4.4. Controlled Gates

To apply gates we split the qubit state into two, based on whether the gate is activated for this pure state (all control qubits are $|1\rangle$). Depending on whether the gate is a

single-qubit or two-qubit gate the corresponding function is called. The resulting state where the gate is activated ($|\psi_A\rangle$ is then summed up with the not-activating state ($|\psi_{NA}\rangle$.

Algorithm 3

function ApplyControlledGate($|\psi_O\rangle, M, C$)
$|\psi_A\rangle \leftarrow (b, \alpha) \in |\psi_O\rangle, \forall c \in C : b[c] = 1$
$|\psi_{NA}\rangle \leftarrow (b, \alpha) \in |\psi_O\rangle, \exists c \in C : b[c] = 0$
$|\psi_A\rangle \leftarrow$ ApplyGate($|\psi_A\rangle$,M, t)
return $|\psi_A\rangle + |\psi_{NA}\rangle$

3.4.5. SWAP

The *SWAP* gate is applied differently depending on whether it is controlled or not. If it is a controlled gate, we apply the gate as two CX gates and one CCX gate. If it is not controlled, we do a "hard" swap. The information we have about one qubit is just exchanged with the information about the other. This way even if one of the qubits is in $\top$, we can still keep our information about the state of the other. If the two qubits are in the same entanglement group, we go through all amplitudes and exchange the value of the two qubits. If they are not in the same entanglement group, we check whether the "internal" index of the qubit in an entanglement has changed and reorder the bits in each basis states as necessary.

3.5. Circuit optimization

This is the heart of the algorithm. Here we use the propagated quantum states to check if gates or control qubits can be removed. For a set of control qubits C, we can find multiple ways to reduce the complexity of the circuit.

3.5.1. Removable Gates

If a gate is never activated, it can be removed. The condition for this is that there is no basis state with a non-zero amplitude for which the gate activates. Or for a qubit state with amplitudes α_b for basis state $|b\rangle$:

$$\forall(\alpha_b, |b\rangle) \in |\psi\rangle : \exists c \in C : b[c] = 0$$

3.5.2. Superfluous Control Qubits

When splitting the qubit state in activating and non-activating, control qubits that are implied by other control qubits do not make a difference. Therefore they can be omitted. c_s can be removed if:

$$\forall(\alpha_b, |b\rangle) \in |\psi\rangle : \bigwedge_{c \in C \setminus c_s} b[c] \implies c_s$$

3.5.3. Algorithm

We define a function that takes the current state of all entanglement groups and a set of controls, and returns whether the gate can be removed and if not, a set of controls that may be smaller but equivalent.

Removing Gates: First we check if the gate can be removed. That is the case iff. it can never activate. To check this we check for every control qubit if the term

$$\forall(b, c) \in |\psi\rangle, b[c] = 0$$

holds. If that is the case we do not look at the rest of the control qubits and just remove the gate.

Control qubits that are $\top$: If we do not know in what state the control qubit is, we have to add the qubit to the returning set of controls.

Control qubits that are $|1\rangle$: If there is a control qubit that is always $|1\rangle$, we can remove it. This is due to the fact that the gate always activates if it is the only control qubit, or it is always implied by the other control qubits as $x \implies 1$ is equivalent to just x.

Finding more implications: Now that we have removed all trivial cases, we can only find implications within an entanglement group. Therefore we group according to the entanglement groups. We then check for each qubit in this group, if there is a pure state where this qubit changes the outcome. If not, it can be removed. This is implemented as shown in algorithm 4. This does not find all implications. If for example $A \iff B \implies C$ and we check B first, it will be removed, leaving A and C as control qubits. Checking for all implications would require exponential run time growing with the number of control qubits. As the benchmark we test the implementation on does not use gates beyond 2 control qubits, we still find all implications there.

3.6. Threshold

After each change of a qubit state, all amplitudes are checked if they are below a certain threshold ε. We argue that when the circuit would be executed on a real quantum

Algorithm 4

function FINDIMPLICATIONS(C, $|\psi\rangle$)
  for $c_s \in C$ **do**
    for $(b, \alpha) \in |\psi\rangle$ **do**
      if $\bigwedge c \in (C \setminus c_s) b[c] \wedge b[c_s] = 0$ **then**
        continue outer loop ▷ c_s changed the outcome (is not implied)
    $C \leftarrow C \setminus c_s$ ▷ c_s was always implied - remove

computer these amplitudes would drown in noise and can therefore be omitted. When dropping amplitudes, the normalization constraint $\sum_{i=0}^{n} |\alpha_i|^2 = 1$ always has to be satisfied. For the old quantum state $|\psi\rangle$ with the amplitudes to keep $\alpha_i, \forall i : |\alpha_i| > \varepsilon$ and the amplitudes to be removed $\alpha_{\leq\varepsilon}, \forall i : |\alpha_{\leq\varepsilon}| \leq \varepsilon$, as well as the reduced quantum state $|\psi_r\rangle$ with the amplitudes α_{i_r}

$$| \, |\Psi\rangle \, | = \sum_{i=0}^{n} |\alpha_i|^2 + |\alpha_{\leq\varepsilon}|^2 = 1 = \sum_{i=0}^{n} |\alpha_{i_r}|^2 = | \, |\Psi_r\rangle \, |$$

To keep the reduced state, where all $\alpha_{\leq\varepsilon}$ were removed normalized, we scale the remaining amplitudes α_i by a factor s

$$| \, |\Psi_r\rangle \, | = \sum_{i=0}^{n} |\alpha_{i_r}|^2 = \sum_{i=0}^{n} \frac{|\alpha_i|^2}{s^2} = 1$$

$$s^2 = \sum_{i=0}^{n} |\alpha_i|^2 = 1 - \sum_{i=0}^{n} |\alpha_{\leq\varepsilon}|^2$$

Which brings the reduced state to

$$|\psi_r\rangle = \sum_{i=0}^{n} \frac{\alpha_i}{s} * |i\rangle = \sum_{i=0}^{n} \frac{\alpha_i \, |i\rangle}{\sqrt{1 - \sum_{i=0}^{n} |\alpha_{\leq\varepsilon}|^2}}$$

If for any reason all amplitudes would be omitted at the same time, the qubit is set to $\top$.

4. Evaluation

All run time results were measured on a laptop with an Intel Core i5-8250U CPU.

4.1. Quantum Circuit Benchmark

In order to evaluate the implementation we use the MQT Bench [12] of the Munich Quantum Toolkit. The aim of this benchmark is to offer a variety of quantum algorithms in order to cover a lot of use cases. Further, algorithms are available in a range of sizes ranging from 2 qubits up to 129.

4.2. Run time

In Figure 4.1, the run times of the `OCAML` and the `C++` implementation are compared. We can see that our implementation achieves a significant speedup for this circuit and parameters. It also shows that the run time does not escalate exponentially when the number of qubits in the circuit increases.

To generalize this, we compare the percentage of decrease in runtime across all circuits in the benchmark. The result is shown in figure 4.2. The orange line shows the median of the speedup. It shows that for $N_{Max} < 4096$ is is about 50% faster. The whiskers show that for some files, this implementation does not work well. The reason for this as well as the reason that for $N_{Max} = 4096$ the median drops down to $\sim 15\%$ needs to be further investigated.

4.3. Reduction of circuits

For most (96%) of the circuits no gates can be removed for $N_{Max} <= 4096$, while at least for most quantum circuits some control qubits can be removed. Removing control qubits in itself helps to reduce the complexity of the quantum circuit as well as making other optimizations such as gate cancellation possible that would otherwise be blocked by the control qubit. The best combination and order to use different tools together needs further research. Two applications where there are gates removed are `shors` algorithm and `qwalk-noancilla`. For shor's algorithm with 18 qubits, around 10% of

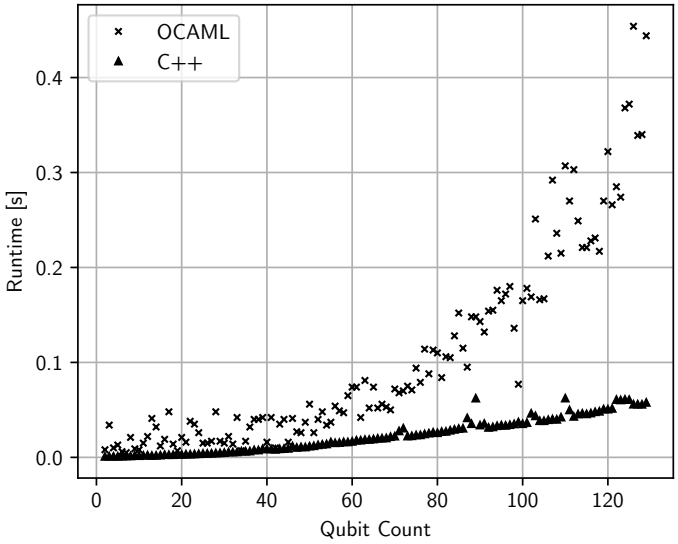

Figure 4.1.: Runtimes of implementations for the wstate circuit $N_{Max} = 2048$, $\varepsilon = 10^{-8}$

its gates are removed with $N_{Max} = 2048$. Across all `qwalk-noancilla` variants 9% of the gates are removed on average.

4.3.1. Maximum number of Amplitudes

We see the effect of changing the maximum number of amplitudes N_{Max} in figure 4.3 and 4.4. The total count summed up over all circuits in the benchmark is shown. We can see that there is a significant increase of the removed gates as well as the removed control qubits when the maximum allowed number of amplitudes is $> 2^6$. This is very interesting as the run time keeps going up with an increasing N_{Max}, which does not seem like it is worth it. At least for the circuits in the benchmark.

4.3.2. Threshold

In figure 4.5 and 4.6 the influence of the threshold on the total count of removed gates and control is show. We can see that the number of removed gates changes very little for a threshold $\leq 10^{-2}$, with the number of removed gates coming to around 900.000. Only when the thresholds is further lifted, the number of removed gates goes up dramatically by about 30 times. Experiments on a real quantum computer would be needed to determine the equality of the results, when compared to the original circuit.

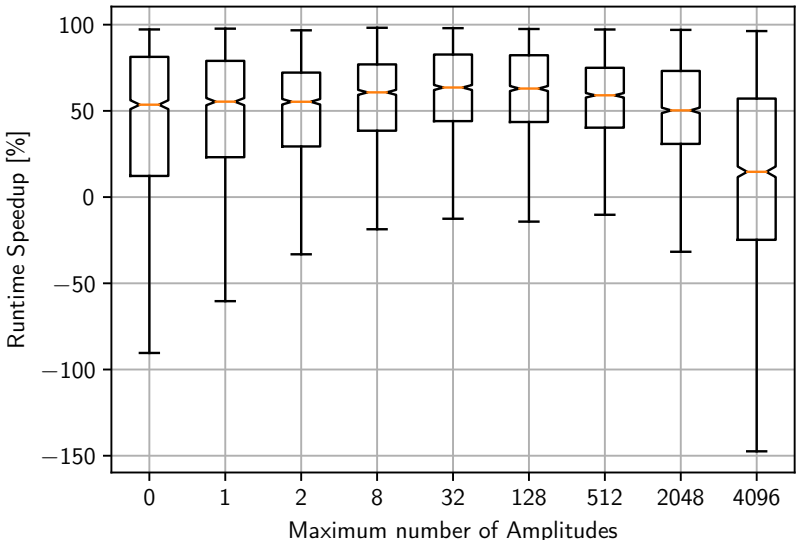

Figure 4.2.: Statistic of speedup by C++ compared to OCAML

If it behaves similar to the threshold when determining the state using a quantum computer, where they only saw a loss of fidelity for a threshold of > 0.2 [8], this would mean that the quantum circuits can be drastically reduced.

The number of removed controls stays relatively constant for all values of the threshold. This might be due to the fact that when the threshold becomes big enough to remove controls, the entire gate is removed and therefore the control not counted in this statistic.

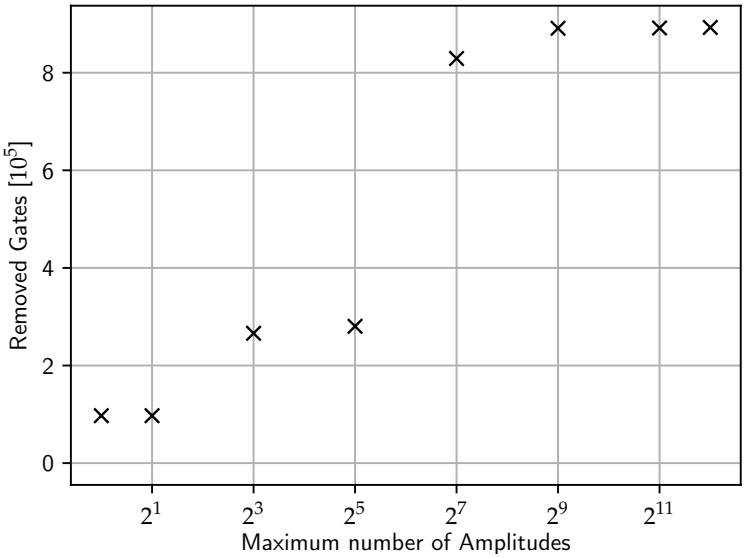

Figure 4.3.: Gates removed over whole benchmark, depending on N_{Max}

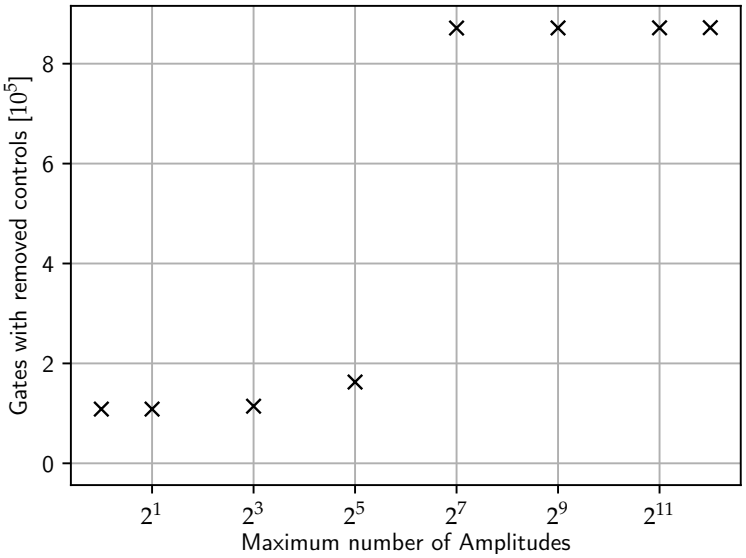

Figure 4.4.: Number of Gates with reduced controls, depending on N_{Max}

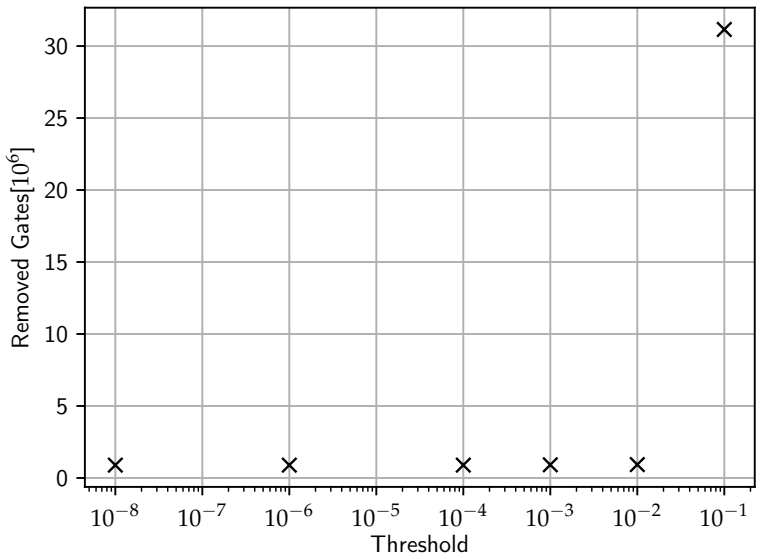

Figure 4.5.: Number of Gates removed, depending on the threshold ε

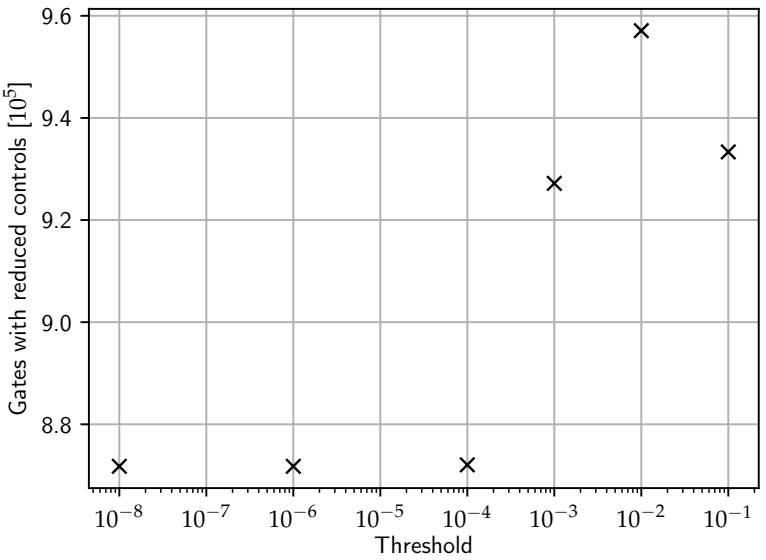

Figure 4.6.: Number of Gates with reduced controls, depending on the threshold ε

5. Outlook

5.1. Run time

Handling complex values in the states and the matrices takes up a lot of the time. One idea to improve that would be to represent complex values as a combination of integer values, and have a look up table of already used complex numbers, as already used to create decision diagrams for quantum circuits [16]. Thereby values can be compared much faster.

Another big potential would be to run entanglement groups on separate threads. Each thread could work until a different qubit is needed and then wait for the thread that works on this qubit. Then they join up and one thread continues working on the now combined group. This would especially work well when there is multiple small entanglement group rather than one large one.

5.2. Circuit Optimization

When two control qubits imply each other, it is currently not defined which will be removed. It would be possible to make this decision dependent on neighbouring gates and which constellation of control qubits enables gate cancellation or other optimizations. This could also extend to more hardware specific optimizations. E.g. focusing on removing not physically neighbouring qubits to reduce later needed *SWAP* gates when the quantum circuit is mapped to the physical quantum computer.

Choosing a threshold ε that works best for a given physical quantum computer and circuit is probably dependent on many variables, such as the error rate of the quantum computer or the size of the circuit.

6. Conclusions

We proposed an algorithm that finds superfluous control qubits and gates in polynomial time on a classical computer. We showed that this version of the algorithm in C++ has a significant run time improvement compared to OCAML. While it does not remove gates in most quantum circuits it does find superfluous control qubits, opening the door for other optimizations. This, as well as the influence of the parameters N_{Max} and ε on different quantum circuits are topics that can be further explored. Especially with respect to hardware specific influences.

List of Figures

List of Tables

Bibliography

[1] M. Backens. "The ZX-calculus Is Complete for Stabilizer Quantum Mechanics." In: *New Journal of Physics*. 9th ser. 16 (2014), p. 093021.

[2] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd. "Quantum Machine Learning." In: *Nature* 549.7671 (Sept. 2017), pp. 195–202. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature23474. arXiv: 1611.09347 [cond-mat, physics:quant-ph, stat].

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

[4] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta. *Open Quantum Assembly Language*. July 13, 2017. arXiv: 1707.03429 [quant-ph]. URL: http://arxiv.org/abs/1707.03429 (visited on 01/20/2023). preprint.

[5] E. C. R. da Rosa and B. G. Taketani. *QSystem: Bitwise Representation for Quantum Circuit Simulations*. Apr. 7, 2020. arXiv: 2004.03560 [quant-ph]. URL: http://arxiv.org/abs/2004.03560 (visited on 04/14/2023). preprint.

[6] L. K. Grover. "A Fast Quantum Mechanical Algorithm for Database Search." In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing - STOC '96*. The Twenty-Eighth Annual ACM Symposium. Philadelphia, Pennsylvania, United States: ACM Press, 1996, pp. 212–219. ISBN: 978-0-89791-785-8. DOI: 10.1145/237814.237866.

[7] J. D. Hidary. *Quantum Computing: An Applied Approach*. Cham, SWITZERLAND: Springer International Publishing AG, 2019. ISBN: 978-3-030-23922-0.

[8] W. Jang, K. Terashi, M. Saito, C. W. Bauer, B. Nachman, Y. Iiyama, R. Okubo, and R. Sawada. "Initial-State Dependent Optimization of Controlled Gate Operations with Quantum Computer." In: *Quantum* 6 (Sept. 8, 2022), p. 798. ISSN: 2521-327X. DOI: 10.22331/q-2022-09-08-798. arXiv: 2209.02322 [quant-ph].

[9] J. Liu, L. Bello, and H. Zhou. "Relaxed Peephole Optimization: A Novel Compiler Optimization for Quantum Circuits." In: *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). Feb. 2021, pp. 301–314. DOI: 10.1109/CGO51591.2021.9370310.

[10] *MQT QFR - A Library for Quantum Functionality Representation Written in C++*. Chair for Design Automation, TU Munich, Apr. 12, 2023.

[11] J. Preskill. "Quantum Computing in the NISQ Era and Beyond." In: *Quantum* 2 (Aug. 6, 2018), p. 79. ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79. arXiv: 1801.00862 [cond-mat, physics:quant-ph].

[12] N. Quetschlich, L. Burgholzer, and R. Wille. *MQT Bench: Benchmarking Software and Design Automation Tools for Quantum Computing*. Sept. 3, 2022. arXiv: 2204.13719 [quant-ph]. URL: http://arxiv.org/abs/2204.13719 (visited on 04/06/2023). preprint.

[13] P. W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer." In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 0097-5397, 1095-7111. DOI: 10.1137/S0097539795293172. arXiv: quant-ph/9508027.

[14] *Std::Vector - Cppreference.Com*. URL: https://en.cppreference.com/w/cpp/container/vector (visited on 05/12/2023).

[15] M. Steffen, J. Chow, S. Sheldon, and D. McClure. *IBM Quantum's Highest Performant System, Yet*. IBM Research Blog. Dec. 21, 2022. URL: https://research.ibm.com/blog/eagle-quantum-error-mitigation (visited on 05/11/2023).

[16] A. Zulehner, S. Hillmich, and R. Wille. "How to Efficiently Handle Complex Values? Implementing Decision Diagrams for Quantum Computing." In: *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). Nov. 2019, pp. 1–7. DOI: 10.1109/ICCAD45719.2019.8942057.

A. Appendix

A.1. Single-Qubit Gates Matrix Representation

Type	Matrix	Type	Matrix
$U3(\theta,\phi,\lambda)$	$U(\theta,\phi,\lambda)$	I	$\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$
$U2(\phi,\lambda)$	$U(\pi/2,\phi,\lambda)$	X	$\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$
$U1(\lambda)$	$U(0,0,\lambda)$	Y	$\begin{pmatrix}0 & -i\\ i & 0\end{pmatrix}$
Z	$\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}$	$P(\phi)$	$U(\phi)$
S	$U(\pi/2)$	$S^\dagger$	$U(-\pi/2$
T	$\begin{pmatrix}1 & 0\\ 0 & e^{\frac{\pi}{4}}\end{pmatrix}$	$T^\dagger$	$\begin{pmatrix}1 & 0\\ 0 & e^{-\frac{\pi}{4}}\end{pmatrix}$
V, SX	$\begin{pmatrix}0,5+0,5i & 0,5-0,5i\\ 0,5-0,5i & 0,5+0,5i\end{pmatrix}$	$V^\dagger, SX^\dagger$	$\begin{pmatrix}0,5-0,5i & 0,5+0,5i\\ 0,5+0,5i & 0,5-0,5i\end{pmatrix}$
$RX(\theta)$	$U(\theta,-\pi/2,\pi/2)$	$RY(\phi)$	$U(\phi,0,0)$
$RZ(\lambda)$	$U(\lambda)$		

Table A.1.: Matrix Definitions for single-qubit gates

A.2. Two-Qubit Gates Matrix Representation

$iSWAP$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & i & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
ECR	$\frac{1}{\sqrt{2}} \begin{pmatrix} 0 & 1 & 0 & i \\ 1 & 0 & -i & 0 \\ 0 & i & 0 & 1 \\ -i & 0 & 1 & 0 \end{pmatrix}$
DCX	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
$RXX(\phi)$	$\begin{pmatrix} \cos(\phi/2) & 0 & 0 & -i\sin(\phi/2) \\ 0 & \cos(\phi/2) & -i\sin(\phi/2) & 0 \\ 0 & -i\sin(\phi/2) & \cos(\phi/2) & 0 \\ -i\sin(\phi/2) & 0 & 0 & \cos(\phi/2) \end{pmatrix}$
$RYY(\phi)$	$\begin{pmatrix} \cos(\phi/2) & 0 & 0 & i\sin(\phi/2) \\ 0 & \cos(\phi/2) & -i\sin(\phi/2) & 0 \\ 0 & -i\sin(\phi/2) & \cos(\phi/2) & 0 \\ i\sin(\phi/2) & 0 & 0 & \cos(\phi/2) \end{pmatrix}$
$RZX(\phi)$	$\begin{pmatrix} \cos(\phi/2) & 0 & -i\sin(\phi/2) & 0 \\ 0 & \cos(\phi/2) & 0 & i\sin(\phi/2) \\ -i\sin(\phi/2) & 0 & \cos(\phi/2) & 0 \\ 0 & i\sin(\phi/2) & 0 & \cos(\phi/2) \end{pmatrix}$
$RZZ(\phi)$	$\begin{pmatrix} \exp(-\phi/2) & 0 & 0 & 0 \\ 0 & \exp(\phi/2) & 0 & 0 \\ 0 & 0 & \exp(\phi/2) & 0 \\ 0 & 0 & 0 & \exp(-\phi/2) \end{pmatrix}$
$XXplusYY(\theta, \beta)$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta/2) & -i\sin(\theta/2)e^{-i\beta} & 0 \\ 0 & -i\sin(\theta/2)e^{i\beta} & \cos(\theta/2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
$XXminusYY(\theta, \beta)$	$\begin{pmatrix} \cos(\theta/2) & 0 & 0 & -i\sin(\theta/2)e^{i\beta} \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -i\sin(\theta/2)e^{-i\beta} & 0 & 0 & \cos(\theta/2) \end{pmatrix}$

Table A.2.: Matrix Definitions for two-qubit gates