Programming Quantum Computers

Krol, A.M.

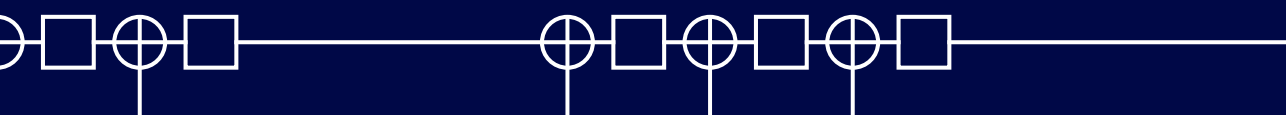
DOI
10.4233/uuid:43110674-5c4d-4745-a941-c8accf328c65

Publication date
2025

Document Version
Final published version

Citation (APA)
Krol, A. M. (2025). *Programming Quantum Computers*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:43110674-5c4d-4745-a941-c8accf328c65

PROGRAMMING QUANTUM COMPUTERS



A M KROL

PROGRAMMING QUANTUM COMPUTERS

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. dr. ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op
donderdag 13 maart 2025 om 10:00 uur

door

Anna Maria KROL

Master of Science in Computer Engineering
Delft University of Technology.

Dit proefschrift is goedgekeurd door de promotoren.

Samenstelling promotiecommissie bestaat uit:

Rector Magnificus,	voorzitter
Prof. dr. H. P. Hofstee,	Technische Universiteit Delft, promotor
Dr. ir. Z. Al-Ars,	Technische Universiteit Delft, promotor

Onafhankelijke leden:

Prof. dr. M.P.M. Möttönen	Aalto Universitet, Finland, and VTT Technical Research Centre of Finland
Prof. dr. G.A. Steele,	Technische Universiteit Delft
Prof. dr. K.L.M. Bertels,	Universiteit Ghent, België
Dr. M. Möller	Technische Universiteit Delft
Dr. rer. nat. A. Luckow	Ludwig-Maximilians-Universität, Germany, Clemson University, USA, and BMW Group, Germany
Prof. dr. Y.M. Blanter	Technische Universiteit Delft, reservelid

ISBN 000-00-0000-000-0

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

I think I can safely say that nobody understands quantum mechanics.

Richard P. Feynman

CONTENTS

SUMMARY

Because of recent stagnating single-thread performance and limited potential for further miniaturization of transistors, the computing industry is looking towards new technologies as the basis for the next generation of computing. One of these new technologies is quantum computing. For utility-scale quantum computing, we will likely need millions of qubits. To program these qubits, the complete quantum computing stack will need to be improved, since programming large numbers of qubits is not feasible with current quantum programming languages.

In this dissertation, we present our new unitary decomposition algorithm, which is used to decompose arbitrary unitary matrices into a sequence of quantum gates that can be executed on a quantum computer. Our method results in 5% less CNOT gates than the previous state-of-the-art and can be used to decompose an arbitrary 3-qubit gate into at most 19 CNOT gates.

Unitary decomposition is an essential part of some quantum algorithms, and can be used as an optimization method for (parts) of quantum circuits. Efficient implementation of unitary decomposition allows for the translation of bigger input matrices into elementary quantum operations, which is key to executing these algorithms on existing quantum computers. With the implementation of unitary decomposition in quantum programming framework OpenQL, we show how the structure of the input or intermediate matrices can be used to minimize the number of output gates and to minimize the runtime of the decomposition. Our implementation is 10 to 500 times as fast as the decomposition methods of the UniversalQCompiler and Qubiter.

With hybrid classical-quantum algorithms, even near-term quantum devices may be able to outperform classical computers. Hybrid algorithms, such as variational quantum eigensolvers, are iterative processes, and use a classical optimizer to update a parameterized quantum circuit. Each iteration, the circuit is executed on a physical quantum processor or a simulator, and the average of the measurement results is passed back to the classical optimizer. When many iterations are performed, the quantum program is recompiled many times.

We have implemented explicit parameters that prevent recompilation of hybrid programs in OpenQL, called OpenQL$_{\text{PC}}$. These parameters reduce the compile time, and therefore improve the total runtime for hybrid algorithms. We have compared the execution of the MAXCUT benchmark in OpenQL with the execution of the same benchmark in PyQuil and Qiskit, which shows that the efficient handling of parameterized circuits in OpenQL$_{\text{PC}}$ results in up to 70% reduction in total compilation time and a reduced total execution time. With OpenQL$_{\text{PC}}$, compilation of hybrid algorithms is also faster than either PyQuil or Qiskit.

In a collaboration with BMW and Entropica, we have developed a quantum algorithm for industrial shift scheduling (QISS), which uses Grover's adaptive search to tackle

a common and important class of valuable, real-world combinatorial optimization problems.

We show how QISS can be used to find the optimal schedule for n days out of a solution space of size $N = 4^{2n}$. The optimal solution is reached in 99% of cases within $\sqrt{N} = 4^n$ applications of Grover's oracle, which requires a total of $11n + 9 + log_2(19n)$ qubits for scheduling n days. We show the explicit construction of the Grover's oracle, incorporating the multiple constraints and detail the corresponding logical-level resource requirements. Further, we simulate the application of QISS for small-scale problem instances to corroborate the performance of the algorithm. Our work shows how complex real-world industrial optimization problems can be formulated in the context of Grover's algorithm.

Using QISS, we then used open-source tools to estimate the quantum resources required for execution of this algorithm. We used qubit models based on current technology, as well as theoretical high-fidelity scenarios for superconducting qubit platforms. We find that the overall computational runtime is more strongly influenced by the execution time of gate and measurement operations than by system error rates. We find that achieving quantum utility would not only require low system error rates (10^{-6} or better), but also measurement operations with an execution time below 10 ns. This rules out the possibility of near-term quantum utility for this use-case, and suggests that significant technological or algorithmic progress will be needed before quantum utility can be achieved.

The research in this dissertation allows us to answer our main research question:
How can we make the quantum computing stack ready for utility-scale quantum computing?
For the quantum stack to be ready for utility-scale quantum computing, several major improvements will need to be made to prepare for programming and compiling circuits with millions of qubits.

- We will need high-level abstractions that will speed up programming of quantum computers, allow for (easier) debugging and will allow for programming millions of qubits.
- The classical component of the compilation and compute of (hybrid) quantum algorithms will need to be improved.
- More algorithms for real-world use-cases will need to be developed, which will provide a basis for improvements across the quantum stack that will lead to quantum utility.
- We need to do quantum resource estimation for real use-cases, in order to have insights into what utility-scale quantum computing will look like.

SAMENVATTING

Vanwege de recente stagnerende single-thread prestaties en het beperkte potentieel voor verdere miniaturisering van transistors, verlegt de computerindustrie haar blik naar nieuwe technologieën als basis voor de volgende generatie computers. Één van deze nieuwe technologieën is de quantum computer. Om quantumvoordeel te bereiken hebben we waarschijnlijk miljoenen qubits nodig. Om deze qubits te programmeren, moet de volledige quantumcomputer-stack worden verbeterd, aangezien het programmeren van grote aantallen qubits niet haalbaar is met de huidige generatie quantumprogrammeertalen.

In dit proefschrift presenteren we ons nieuwe unitaire decompositie algoritme, dat kan worden gebruikt om willekeurige unitaire matrices te factoriseren tot een reeks quantum operaties die kunnen worden uitgevoerd op een quantumcomputer. Onze methode resulteert in 5% minder CNOTs dan de state-of-the-art en kan worden gebruikt om een willekeurige 3-qubit operatie te vertalen naar maximaal 19 CNOTs.

Unitaire decompositie is een essentieel onderdeel van sommige quantumalgoritmes en kan worden gebruikt als een optimalisatiemethode voor (delen van) quantumcircuits. Efficiënte implementatie van unitaire decompositie maakt de vertaling van grotere matrices naar elementaire quantum operaties mogelijk, wat essentieel is voor het uitvoeren van deze algoritmes op bestaande quantumcomputers. Met de implementatie van unitaire decompositie in het quantumprogrammeerframework OpenQL laten we zien hoe de structuur van de ingevoerde matrix of tussenmatrices kan worden gebruikt om het aantal uitvoer operaties te minimaliseren en de tijd die de decompositie kost te minimaliseren. Onze implementatie is 10 tot 500 keer zo snel als de decompositiemethoden van de UniversalQCompiler en Qubiter.

Met hybride klassieke-quantumalgoritmes kunnen zelfs quantumcomputers op korte termijn klassieke computers overtreffen. Hybride algoritmes, zoals variationele quantumeigensolvers, zijn iteratieve processen en gebruiken een klassieke optimalisator om een geparameteriseerd quantumcircuit elke iteratie bij te werken. Bij elke iteratie wordt het circuit uitgevoerd op een fysieke quantumprocessor of een simulator, en het gemiddelde van de meetresultaten wordt teruggestuurd naar de klassieke optimalisator. Wanneer er veel iteraties worden uitgevoerd, wordt het quantumprogramma vele malen opnieuw gecompileerd.

We hebben expliciete parameters geïmplementeerd die hercompilatie van hybride programma's in OpenQL voorkomen, genaamd OpenQL$_{\text{PC}}$. Deze parameters verminderen de compileertijd en verbeteren daarom de totale tijd die de uitvoer van een hybride algoritme kost. We hebben de uitvoering van de MAXCUT-benchmark in OpenQL vergeleken met de uitvoering van dezelfde benchmark in PyQuil en Qiskit, wat aantoont dat de efficiënte verwerking van geparametriseerde circuits in OpenQL$_{\text{PC}}$ resulteert in tot 70% vermindering van de totale compilatietijd en een kortere totale uitvoeringstijd. Met OpenQL$_{\text{PC}}$is de compilatie van hybride algoritmes ook sneller dan in PyQuil of Qiskit.

In samenwerking met BMW en Entropica hebben we een quantumalgoritme voor industriële planningen (QISS) ontwikkeld, dat gebruik maakt van Grover's adaptieve zoekalgoritme om een veelvoorkomende en belangrijke klasse van waardevolle, echte combinatorische optimalisatieproblemen aan te pakken.

We laten zien hoe QISS gebruikt kan worden om de optimale planning te vinden voor n dagen uit een oplossingsruimte die $N = 4^{2n}$ elementen bevat. De optimale oplossing wordt bereikt in 99% van de gevallen met minder dan $\sqrt{N} = 4^n$ applicaties van Grover's orakel, wat een totaal aan $11n+9+log_2(19n)$ qubits gebruikt voor het plannen van n dagen. We laten de expliciete constructie van het Grover's orakel zien waarbij de meerdere restricties worden opgenomen, en hoe dat vertaalt naar de behoeftes van het orakel op het niveau van logische qubits en operaties. Verder simuleren we de toepassingen van QISS voor kleinschalige problemen om de prestaties van het algoritme te bevestigen. Ons werk laat zien hoe complexe industriële optimalisatieproblemen kunnen worden geformuleerd in de context van Grover's algoritme.

Met behulp van QISS hebben we vervolgens open-source applicaties gebruikt om een inschatting te maken van het aantal qubits die nodig zijn en van de uitvoeringstijd van dit algoritme. We hebben qubitmodellen gebruikt op basis van huidige technologie, evenals theoretische hoge-kwaliteits scenario's voor supergeleidende qubits. We hebben ontdekt dat de algehele rekentijd sterker wordt beïnvloed door de uitvoeringstijd van quantumoperaties dan door qubit error-percentages. Onze bevindingen laten zien dat het behalen van quantumvoordeel niet alleen lage qubit errorpercentages (10^{-6} of beter) vereist, maar ook quantummetingen die uitgevoerd kunnen worden in minder dan 10 ns. Dit sluit uit dat quantumvoordeel op korte termijn voor dit gebruikersscenario mogelijk is en laat zien dat er aanzienlijke vooruitgang nodig is in quantumtechnologiëen- of algoritmes, voordat er voor dit scenario quantumvoordeel kan worden behaald.

Dit brengt ons terug bij onze belangrijkste onderzoeksvraag: **Hoe kunnen we de quantumcomputerstack gereedmaken voor quantumcomputers op bruikbare schaal?**
Om de quantumstack gereed te maken voor quantumcomputers op bruikbare schaal, zullen er verschillende grote verbeteringen moeten worden ingevoerd om het programmeren en compileren van circuits met miljoenen qubits mogelijk te maken.

- We hebben abstracties op hoog niveau nodig die het programmeren van quantumcomputers kunnen versnellen, die (eenvoudiger) debuggen mogelijk maken en die het programmeren van miljoenen qubits mogelijk maken.
- Het klassieke onderdeel van de compilatie en berekening van (hybride) quantumalgoritmes moet worden verbeterd.
- Er moeten meer algoritmes voor echte gebruikersscenario's worden ontwikkeld, die een basis vormen voor verbeteringen in de quantumstack die zullen leiden tot het (grootschalig) gebruik van quantum computers.
- We moeten inschattingen maken van de (quantum)benodigdheden voor echte gebruikersscenario's, om inzicht te krijgen in hoe quantumcomputers op bruikbare schaal eruit zal gaan zien.

1

INTRODUCTION

This thesis is about trying to improve quantum programming, from the fundamental, compilation and implementation levels, ending with a look to the future of quantum computing.

1.1. A BRIEF INTRODUCTION TO QUANTUM COMPUTING

This section introduces why people are looking at quantum computing, how to build a quantum computer, the current state of quantum computer and ends with a look to the future.

1.1.1. WHY QUANTUM COMPUTING?

Computers have seen a steady exponential increase in performance since the 1960s. As Gordon Moore observed in the 1970, the number of components per integrated circuit doubled every two years. This trend, called Moore's law, has held until the 2010s but now, in 2024, has ended [202].

At the heart of Moore's law has been the miniaturization of transistors [121], from 20μm in 1968 to just 3nm in 2022 [122]. But there is limited potential for making even smaller transistors: for a 3nm transistor, the gate length is only 15 silicon atoms across [89]. Besides the fundamental limits of silicon atoms, the potential for further performance increase is also limited by power consumption, heat dissipation, data storage and memory access time [207].

To counteract stagnating single-thread performance, the industry has shifted towards parallel computing using multiple cores (Figure 1.1) and off-loading certain tasks to dedicated hardware, such as GPUs, FPGAs or, in the future, quantum computers [135].

1.1.2. HOW TO BUILD A QUANTUM COMPUTER?

The fundamental units of a quantum computer, qubits, consist of one elementary particle each. Depending on the technology, this can be an atom, an ion, an electron or a photon.

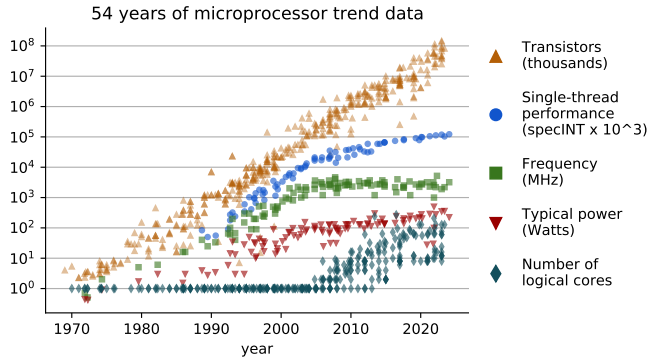


Figure 1.1: 54 years of microprocessor trend data. Original data up to 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten, data 2010-2020 collected by K. Rupp [163, 164], additional data (1970-2024) via wikipedia.org/wiki/Transistor_count [201] and the SPECint and SPECpower benchmarks [42].

Qubits are very susceptible to noise, in the form of electromagnetic radiation, temperature, imperfections in the control of the qubits (quantum gates and measurement) or other interactions with the environment [90]. Depending on the technology used to make qubits, this means that they need to be held in a vacuum, at a temperature near absolute zero, and generally isolated and protected from the environment. But some access is needed to be able to control and measure the qubit and to allow qubits to interact with each other in controlled ways.

The DiVincenzo criteria to build a quantum computer are [54]:

1. A scalable physical system with well-characterized qubits
2. The ability to initialize the qubits to a simple reference state (such as $|000....\rangle$)
3. Long relevant coherence times, much longer than the gate operation time
4. A universal set of quantum gates
5. A qubit-specific measurement capability

To fulfill these requirements, you do not just need qubits but also the systems to control qubits. For controlling qubits at scale, you need to be able to program the quantum computer, which is done using a compiler stack like the one shown in Figure 1.2.

1.2. CURRENT STATE OF QUANTUM COMPUTING

The technology of current qubits does not yet fulfill all of the DiVincenzo criteria to a point where they can be used to do things that are infeasible with any classical computer.

Although the first systems with more than a thousand qubits are available [34, 157], the error rates of these qubits and the quantum gates are too high to do anything more than short experiments before any information in the systems is lost due to noise.

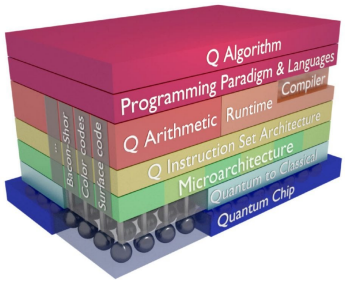


Figure 1.2: The quantum computing stack

This is why error correction methods are being developed, which means multiple physical qubits can be used to make a single logical qubit. One such method is surface code, which has a threshold of 0.57% error rate of the physical qubits [61, 30]. This threshold gives the maximum error rate that the physical qubits can have, below which error correction works to prevent errors in the system. With error rates near the threshold, many qubits are needed per physical qubit, on the order of 10 000 or more. If the physical error rate of the qubits is much lower than the threshold, fewer physical qubits are needed per logical qubit.

Different quantum error correction codes (QEC) have different thresholds, and the threshold can also be different for different types of errors. Current quantum systems are butting up against this error threshold [2], but we are not yet in the fault-tolerant era of quantum computing without sufficient physical qubits to fully implement QEC.

The error rates have been decreasing, and qubit numbers increasing in recent years (see Figure 6.1).

1.3. THE FUTURE OF QUANTUM COMPUTING

The computational properties of quantum computers allow them to solve certain types of problems with (super)polynomial speedup as compared to classical computers. We will need a quantum computer with low enough error rates to enable quantum error correction, and with many thousands or potentially millions of qubits, most of which will be used for error correction [22].

1.3.1. QUANTUM SUPREMACY, UTILITY AND ADVANTAGE

Three different terms are often used when talking about milestones in the development of quantum computing: *quantum supremacy*, *quantum advantage* and *quantum utility*. There is no standardized definition of these concepts, and they are used to mean different, sometimes conflicting, things by different authors and in different contexts. All three concepts are used when talking about how quantum computing might be better than classical computing in some way: when quantum computing has a *supremacy*, *utility* or *advantage* over classical computing.

Because of the conflicting definitions of these three concepts, we will define these terms here as they are used in this thesis:

- **Quantum supremacy**: a quantum computer solves a problem that cannot be solved by any classical computing system, regardless of the utility of the problem [152].
- **Quantum utility**: quantum computing has become a useful tool for solving meaningful (scientific) problems that are beyond the reach of brute-force classical computing methods [106, 47].
- **Quantum advantage**: a quantum computer can solve a problem with some (significant, practical) benefit over a classical computer; the problem needs to have a real-world application and be commercially or scientifically relevant, the benefit can be a decrease in runtime, but quantum advantage also applies for other types of benefits, like cost or accuracy [68, 47]. Often overlaps with the meaning of quantum utility: both terms were coined to contrast with *quantum supremacy*.

For quantum utility and quantum advantage, we do not just need quantum supremacy, but we need to solve a useful problem that a quantum computer can solve in a reasonable amount of time, such as hours or weeks, rather than millions of years.

1.3.2. SHOR'S AND GROVER'S ALGORITHMS

The threshold for quantum utility and quantum advantage depends greatly on the type of problem and what classical and quantum algorithms can be used to solve the problem. This can clearly be seen in Figure 1.3.

A brute force approach can be used to solve most types of problems, but such an approach is generally not practically feasible due to the large amount of compute resources required. However, in some cases the brute force approach is the only possibility, which is why it is used in exhaustive search subroutines for solving NP-complete problems [8], for example. With quantum computing, Grover's algorithm can be used to solve such problems with polynomial speedup: instead of $O(2^L)$ function evaluations, only $O(\sqrt{2^L})$ are needed to search a space of size 2^L.

As can be seen in Figure 1.3, this does result in a quantum speedup compared to classical brute force for larger problem sizes. Even at a frequency of 1 MHz (10^6 function evaluations per second), the crossover point is below one minute of runtime compared to a classical computing system running at (the equivalent of) 7.7 THz ($7.7 \cdot 10^{12}$ function evaluations per second). But even though it is faster than the classical computing approach, the runtime for Grover's algorithm increases rapidly with problem size. This means that there is a limited range of problem sizes for which Grover's algorithm will be useful. If we set the limit for reasonable runtime at a week, then the maximum search space for the classical computer is 2^{62}, while for a quantum computer at 1 MHz it is 2^{78}. This will limit the usefulness of Grover's algorithm. For faster quantum computers, the window of runtime speedup is larger, especially if a search algorithm that achieves cubic or quartic speedup is found [14].

Compared to Grover's algorithm, Shor's algorithm has a much narrower set of problems for which it can be used. One such problem is factoring, which can be solved using a brute-force approach, but can also be solved using a more efficient algorithm: the general number field sieve. This algorithm has a complexity of

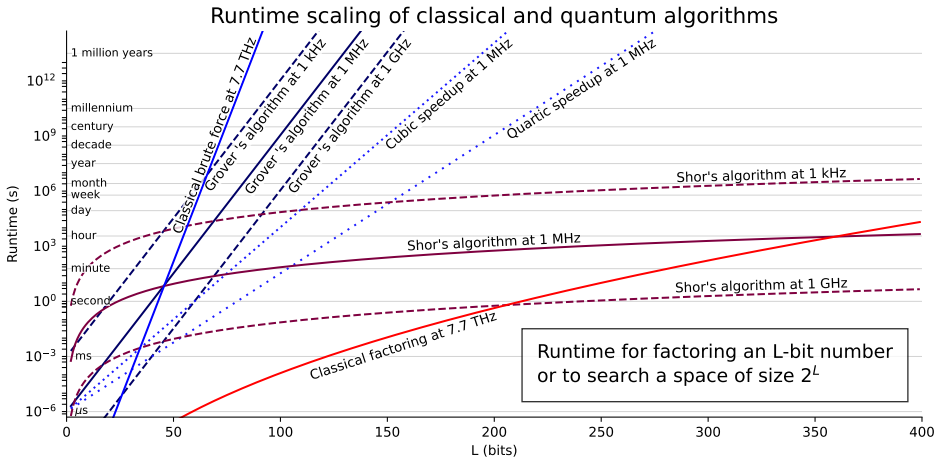


Figure 1.3: Classical and potential quantum runtime scaling for factoring of an L-bit number or for searching an unstructured search-space with 2^L items. The classical computing frequency of 7.7 THz is the effective speed for factoring of the largest RSA number to date, which has 829 (binary) digits and took the equivalent of 2700 core-year using Intel Xeon Gold 6130 CPUs at 2.1GHz [26].

$O\left(\exp\left(1.9\cdot\sqrt[3]{L\cdot\log(L)^2}\right)\right)$ [27] and was used to factor the largest RSA number to date, a number with 250 decimal digits or 829 binary digits. Shor's algorithm uses $2L^2$ logical qubits and $O\left(72\cdot L^3\right)$ logical cycles to factor an L-bit number [21]. And even though Shor's algorithm is slower for smaller problem sizes, factoring numbers with more than 360 bits will be quicker when using a quantum computer at 1 MHz compared to the classical systems.

The crossover point between the quantum and classical approaches depends on many different factors, from the algorithm implementation to the characteristics of the physical qubits [193]. The difficulty in determining the exact problem size at which the quantum computer will have an advantage over the classical system can clearly be seen in the Figure 1.3. When the quantum computer operates at a (logical) frequency of 1 GHz, the maximum size of a feasible search space with Grover's algorithm is not 2^{78}, but 2^{98}, and Shor's algorithm is already faster for factoring 206-bit numbers. For quantum computing at 1 kHz, however, a classical computer at a runtime of one week can search a larger space than a quantum computer using Grover's, and the crossover point for Shor's algorithm occurs at the factoring of a 562-bit number, which would take a year. And this is without taking into account the potential for parallel execution of the classical computers. The number of 7.7 THz already accounts for some parallel execution, but while factoring the largest RSA number took the *equivalent of* 2700 core-years, it did not actually take 2700 years. This might be even more the case for the type of brute-force approaches that Grover's algorithm can replace, because an unstructured search can easily be split into many separate search tasks that can all be executed in parallel.

The figure also show the difference between the two algorithms. Grover's algorithm can be used for a wide range of problems and provides quadratic speedup over brute-

force methods. But a practical speedup can only be achieved by an (unrealistically) fast quantum computer, and only for a limited set of problem sizes, because the runtime of the quantum computer will still grow exponentially with size. The type of brute-force (unstructured search) problem that Grover's can be used for, is also a problem class that can be easily parallelized in classical computing (embarrassingly parallel), which means that the brute-force execution time can be reduced to (almost) the time it takes for one single iteration. This is not possible with Grover's algorithm. All this combined means that it is unlikely there will many practical applications for Grover's algorithm in the near to medium future. However, if a quantum computer is developed with a high enough capacity and clock speed, then it can be used to speed up many different types of applications using Grover's algorithm.

This is in contrast with Shor's algorithm. Shor's algorithm has a much narrower field of application than Grover's, and can be used to solve a much more specific type of problem: factoring [177][1]. And although factoring can also be done using classical computing at below exponential runtimes, factoring large numbers still takes a lot of time and a lot of compute resources. This is the reason that the widely used RSA-encryption works as a cryptosystem: the largest RSA-key that has currently been broken has 829 bits [26], while typical key sizes used for encryption are 2048-4096 bits. With Shor's algorithm, factoring can be done at superpolynomial speedup compared to all known non-quantum algorithms [177], and decryption of RSA-keys might become possible [193], which both provides a practical application for Shor's algorithm and incentive for building a quantum computer that is powerful enough to use Shor's to break RSA-encrypted data.

This shows that the threshold for quantum utility depends not only on the achievable clock-speed of the quantum computer, but also on the type of algorithm that is used, and how that compares to a classical solution to the problem.

For both Shor's and Grover's algorithm, many qubits are needed to solve the problem at (and beyond) the crossover point, especially if quantum error correction and other overheads need to be included.

1.4. CHALLENGES TO ADDRESS AND RESEARCH QUESTIONS

For quantum utility, we do not just need to have enough qubits, we need to program these qubits. The overarching research question of this thesis is thus:

- How can we make the quantum computing stack ready for utility-scale quantum computing?

To answer this question, this dissertation takes a broad look at the whole compiler stack, as shown in Figure 1.2.

We will address the following problems with the current quantum programming stack:

1. Limited number of high-level (quantum specific) abstractions

2. Classical compile/compute time, especially in hybrid algorithms

[1] Or more widely: the hidden subgroup problem for finite abelian groups

3. Limited number of algorithms for real-world use-cases
4. Lack of visibility on quantum computing effectiveness

1.4.1. QUANTUM HIGH-LEVEL ABSTRACTIONS

In order to perform computations using a quantum computer, we need to manipulate qubits using quantum gates and measurement operators. These qubit manipulations represent the lowest level of modifications in the physical characteristics of the qubits. Such modifications do not relate directly to the algorithms being executed on the quantum computer itself. In that sense, such qubit manipulations correspond to programming a classical computing system using assembly language. However, in modern programming languages used to program classical computers, programmers no longer need to program in assembly language but can make use of higher-level abstractions, such as functions, structs and if-then-else statements, that simplify the task of programming, optimizing and debugging algorithms on such computers.

Current quantum programming languages usually allow the use of high-level classical programming features, but there are not many quantum-specific high-level abstractions that can be used. This is partly because there is no immediate need for high-level abstractions when programming current quantum computers. The number of qubits on a single quantum chip is still quite manageable to program without high-level abstractions, and hand-optimized quantum circuits are at an advantage over potentially less efficient automated code. Quantum programming will also need to mature as a field before a general consensus can form about the type and function of high-level abstractions.

One existing high-level abstraction is unitary decomposition, with which a (unitary) matrix can be used directly when programming a quantum circuit. This matrix needs to be decomposed into elementary or native gates.

This brings us to the following two research questions about quantum high-level abstractions:

1. **What are useful high-level abstractions for programming quantum computers?**
2. **How can existing high-level abstractions such as unitary decomposition be improved?**

1.4.2. CLASSICAL COMPILE AND COMPUTE TIME

The experimental nature of most current quantum programs also means that there is little regard for the classical compute and compile time of algorithms. This is an issue when the goal is to minimize the total runtime of the algorithm. Minimizing classical components of the runtime becomes more relevant for larger scale quantum computing, first for outperforming classical computers at some task (quantum supremacy) but especially when using quantum computing for actual (commercial) purposes.

This brings us to the following research question to address the classical compile and compute time in quantum computing:

3. **Can the performance of quantum algorithms be improved through compiler optimizations?**

1.4.3. REAL-WORLD USE-CASES

Quantum computers of the future can be used to solve large-scale optimization problems, like the supply chain of a large company. Although current quantum computers cannot be used yet for large-scale problems, it is valuable for the development process to keep certain end-goals in mind. One way to do that is by using benchmarks to compare different quantum computers, where the problems used in the benchmark are representative of industry-relevant problems. This will also aid with the development of a quantum compiler stack that will be suitable for writing algorithms in the future.

This brings us to the following research question to address the limited number of algorithms for real-world use-cases:

4. **What does a quantum algorithm for a real-world use-case look like?**

1.4.4. REQUIREMENTS FOR PRACTICAL QUANTUM COMPUTING

To make the quantum computing stack ready for utility-scale quantum computing, we need to know what that scale is. How many qubits will be needed to be able to run useful quantum algorithms?

Besides the number of physical qubits, the viability of quantum computers also depends on the physical and logical clock speed of the qubits, the error rate of idle qubits, the error rate of qubit operations, qubit interconnect topology, available error correction codes and other details of the qubit implementation and technology [193]. And the scale at which a quantum computer can outperform a classical computer also depends on the type of problem we want to solve, and the specifics of the algorithm and implementation [193].

This brings us to the following research question to address the lack of visibility on quantum computing effectiveness:

5. **What are the requirements for practical quantum computing?**

1.5. CONTRIBUTIONS

In this section, we present the contributions of this dissertation for each of the research questions posed in Section 1.4.

1. What are useful high-level abstractions for programming quantum computers?

- Chapter 2: High-level circuit blocks are used in the construction of a new unitary decomposition algorithm.
- Chapter 3: The structure and construction of different decomposition algorithms are discussed, including the other uses these algorithm may have, such as state preparation.
- Chapter 4: Hybrid programming constructions are used, optimized and compared between different quantum programming languages.
- Chapter 5: We use high-level building blocks to construct our algorithm. We show the explicit construction of each block and how the problem constraints were used

to build up the algorithm. Using these building blocks, quantum oracles for many different types of optimization problems can be created.

2. How can existing high-level abstractions such as unitary decomposition be improved?

- Chapter 2: We present a new unitary decomposition algorithm that results in 5% less CNOT gates than the state-of-the-art best algorithm.
- Chapter 3: We show we can take advantage of the underlying matrix structure to reduce the size of the circuit resulting from unitary decomposition.

3. Can the performance of quantum algorithms be improved through compiler optimizations?

- Chapter 3: We accelerate the compilation of our implementation of unitary decomposition in quantum programming framework OpenQL. Our implementation is up to 500 times as fast as other implementations.
- Chapter 4: We implement explicit parameters that prevent recompilation of the whole quantum circuit. This improves compilation and thereby the total runtime for iterative hybrid quantum-classical algorithms. This reduces compile time by up to 70% for the MAXCUT benchmark.

4. What does a quantum algorithm for a real-world use-case look like?

- Chapter 5: We present QISS, a Quantum algorithm for Industrial Shift Scheduling. This algorithm uses Grover's adaptive search to find the optimal combination of shift lengths for a simplified automotive production line. The structure and constraints of the problem are modeled after a real-world use-case. The algorithm was developed in a collaboration with BMW and Entropica Labs due to the industrial impact of such algorithm on optimizing complex manufacturing processes.

5. What are the requirements for practical quantum computing?

- Chapter 6: Using QISS, we estimate the quantum resources required to execute the algorithm for a range of problem sizes. We use different qubit characteristics as the basis of our resource estimations: characteristics from the literature, characteristics based on current superconducting qubits and potential characteristics of high-fidelity superconducting qubits. We show that we will need hundreds of thousands of qubits with low error rates and fast logical cycle times to achieve quantum utility for this application.

1.6. OUTLINE

In this introduction, we have explained the promises and challenges of quantum computing. We have outlined our contributions to these challenges. The remainder of the dissertation describes the algorithms, optimizations and our vision in more detail.

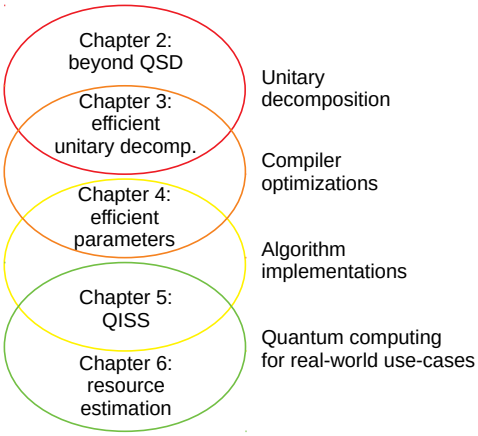


Figure 1.4: Relation between the thesis chapters

Figure 1.4 shows the order of and the relationships between the chapters of the dissertation. The first two chapters, Chapter 2 and Chapter 3, focus on two aspects of unitary decomposition. In Chapter 2, we discuss our new decomposition algorithm, and Chapter 3 shows our earlier implementation and optimization of the quantum Shannon decomposition algorithm. Chapters 3 and 4 both focus on compiler optimizations of programming framework OpenQL. The compiler optimization in Chapter 4 is verified and tested by implementation of the MAXCUT benchmark. The last two chapters are the result of a collaboration with BMW, Germany and Entropica labs, Singapore: The first of these, Chapter 5, discusses the development of our own benchmark, Quantum Industrial Shift Scheduling algorithm (QISS), which is based on a real-world use-case from BWM. This benchmark is used to assess the requirements for industrial quantum computing through resource estimation, which is discussed in Chapter 6.

2

BEYOND QUANTUM SHANNON: CIRCUIT CONSTRUCTION FOR N-QUBIT GATES

In this chapter, we present our new unitary decomposition algorithm, which is a fundamental improvement of the unitary decomposition abstraction for quantum programming. The new algorithm is based on the block-ZXZ decomposition method and can be used to decompose arbitrary unitary matrices into a sequence of quantum gates that can be executed on a quantum computer.

Our method results in 5% less CNOT gates than the previous state-of-the-art: quantum Shannon decomposition, which was first introduced in 2005 by Möttönen and Vartiainen. The decomposition is applied recursively to generic quantum gates, and can take advantage of existing and future small-circuit optimizations. Because our method uses only single qubit gates and uniformly controlled rotation-Z gates, it can easily be adapted to use other types of multi-qubit gates.

With the proposed decomposition, a general three-qubit gate can be decomposed using 19 CNOT gates (rather than 20). For general n-qubit gates, the proposed decomposition generates circuits that have $\frac{22}{48}4^n - \frac{3}{2}2^n + \frac{5}{3}$ CNOT gates, which is less than the best known exact decomposition algorithm by $(4^{n-2}-1)/3$ CNOT gates. This chapter has the following contributions:

- We show how to decompose an arbitrary n-qubit gate into at most $\frac{22}{48}4^n - \frac{3}{2}2^n + \frac{5}{3}$ CNOT gates. This is $(4^{n-2}-1)/3$ less than the best previously published work [173].

- More specifically, we can construct a general three-qubit operator with at most 19 qubits, which is currently the least known for any exact decomposition method.

This chapter is based on the following article:

- Anna M. Krol and Zaid Al-Ars. "Beyond quantum Shannon decomposition: Circuit construction for n-qubit gates based on block-ZXZ decomposition". In: *Phys. Rev. Appl.* 22 (3 Sept. 2024), p. 034019. DOI: 10.1103/PhysRevApplied.22.034019

CODE AVAILABILITY

An implementation of the algorithm using Python and Qiskit can be found here: https://github.com/anneriet/decomposition_algorithm.

2.1. INTRODUCTION

To execute a quantum algorithm, a series of unitary operations (gates) and non-unitary operations (measurements) are applied to quantum bits (qubits) in a quantum circuit. The complexity of a quantum algorithm can be described as the number of gates, the number of qubits or the length of the critical path (depth) of the circuit.

Physically, a qubit is a quantum-mechanical system that can store quantum information, such as superconducting qubits [110], trapped ions [78] or spin qubits [134]. Applying a quantum gate means manipulating the state of the qubit in a controlled way. Exactly which gate operations are possible depends on the qubit technology and the implementation [123].

To run arbitrary quantum operations on real quantum hardware, the unitary operator (matrix) needs to be translated into elementary (native) gate operations. This is by no means a trivial task, and the focus of much research over the years into methods for performing such translation using quantum gate decomposition.

An important target of gate decomposition methods is to minimize the number of two-qubit gates required to implement a given unitary matrix. This is essential, because the two-qubit gates require qubit connectivity and mapping, and the execution time and error-rates of two-qubit gates are an order of magnitude worse than for single qubit gates in current quantum hardware [123].

It has been proven that any exact decomposition of an arbitrary n-qubit gate requires at least $\frac{1}{4}(4^n - 3n - 1)$ CNOT gates [176].

Approximate decomposition algorithms such as [156, 12, 205] can be used to decompose arbitrary quantum gates with (almost) the minimum number of CNOT gates and little accuracy loss, at the cost of excessive runtime of the search algorithm: decomposition of a five-qubit gate can take at least several hours. These methods are therefore not suitable for bigger gates or for applications where classical compile time is relevant for the performance of the algorithm [117].

In contrast, exact decomposition methods are much faster, and for one- and two-qubit gates also achieve the minimum CNOT count. One-qubit gates do not require any CNOTs and can be decomposed into a sequence of three rotation gates [19]. Arbitrary two-qubit gates can be decomposed into three CNOTs using the methods described in [196, 176, 198, 182], which also show that less CNOTs are necessary when the gate meets certain conditions. For arbitrary three-qubit gates, there is no algorithm that results in the minimum 14 CNOTs, but algorithms do exist that can decompose them into 64 [194], 40 [195], 26 [80] or 20 [139, 173] CNOTs.

For quantum gates of arbitrary size, the decomposition methods have drastically improved since 1995, when Barenco et al. [19] showed that any unitary operator on n qubits can be constructed using at most $O(n^3 4^n)$ two-qubit gates. This decomposition method used the standard QR decomposition based on Givens rotations [44], and the CNOT count has been improved over the years by use of Gray codes and gate cancellations to $O(\frac{1}{2} \cdot 4^n)$ CNOT gates [3, 173, 194]. Another approach to unitary decomposition has been to use Cosine Sine Decomposition (CSD) [147, 69, 191, 140]. This was combined with diagonalization and separate handling of quantum multiplexors (using the method from [140]) in 2004 to construct the NQ decomposition, which requires $O(\frac{1}{2} \cdot 4^n)$ CNOTs [175]. The NQ decomposition was optimized in 2005 by Möttönen and Varti-

ainen to produce the first decomposition with less than $\frac{1}{2}4^n$ in the leading order: with the optimizations, the decomposition requires at most $(23/48)\cdot 4^n-(3/2)\cdot 2^n+(4/3)$ CNOT gates [139]. This decomposition is more widely known as the Quantum Shannon decomposition (QSD) [173]. More recently, the Khaneja-Glaser decomposition [104] was used in [129] to construct a decomposition method that can decompose unitary operations using $(21/16)\cdot 4^n-3(n\cdot 2^{n-2}+2^n)$ CNOT gates.

In this chapter, we show the design and construction of a new unitary decomposition method based on block-ZXZ decomposition [50, 49, 62], that uses demultiplexing and optimizations similar to quantum Shannon decomposition [139, 173]. The contributions of this chapter are as follows.

- We show how to decompose an arbitrary n-qubit gate into at most $(22/48)\cdot 4^n-(3/2)\cdot 2^n+(5/3)$ CNOT gates. This is $(4^{n-2}-1)/3$ less than the best previously published work [139, 173].
- More specifically, we can construct a general three-qubit operator with at most 19 qubits, which is currently the least known for any exact decomposition method.

An overview of the CNOT count for the proposed method compared to previously published unitary decomposition algorithms is given in Table 2.1.

Table 2.1: Number of CNOT gates resulting from unitary decomposition by the proposed decomposition compared to previously published algorithms and the theoretical lower bound. The results of this chapter are shown in bold.

Number of qubits	1	2	3	4	5	6	n
Original QR decomp. [19, 44]							$O(n^3\cdot 4^n)$
Improved QR decomp. [112]							$O(n\cdot 4^n)$
Palindrome transform [3, 173]							$O(n\cdot 4^n)$
Givens rotations (QR) [194]	0	4	64	536	4156	22618	$\approx 8.7\cdot 4^n$
Original CSD [191, 118]	0	14	92	504	2544	12256	$(1/2)\cdot n\cdot 4^n-(1/2)\cdot 2^n$
Iterative disentangling (QR) [173]	0	8	62	344	1642	7244	$2\cdot 4^n-(2n+3)\cdot 2^n+2^n$
KG Cartan decomp. [129]	0	3	42	240	1128	4896	$(21/16)\cdot 4^n-3(n\cdot 2^{n-2}+2^n)$
CSD [140]	0	8	48	224	960	3968	$4^n-2\cdot 2^n$
QSD (base) [173]	0	6	36	168	720	2976	$(3/4)\cdot 4^n-(3/2)\cdot 2^n$
Block-ZXZ [50]	0	6	36	168	720	2976	$(3/4)\cdot 4^n-(3/2)\cdot 2^n$
CSD (optimized) [139]	0	4	26	118	494	2014	$(1/2)\cdot 4^n-(1/2)\cdot 2^n-2$
NQ [175]	0	3	21	105	465	1953	$(1/2)\cdot 4^n-(3/2)\cdot 2^n+1$
QSD (optimized) [139, 173]	0	3	20	100	444	1868	$(23/48)\cdot 4^n-(3/2)\cdot 2^n+(4/3)$
Proposed decomposition	**0**	**3**	**19**	**95**	**423**	**1783**	$\mathbf{(22/48)\cdot 4^n-(3/2)\cdot 2^n+(5/3)}$
Theoretical lower bounds	0	3	14	61	252	1020	$(1/4)\cdot(4^n-3n-1)$

The rest of the chapter is organized as follows. We start with the notation and gate definitions in Section 2.2. Then in Section 2.3, we show the decomposition of uniformly controlled rotations. Section 2.4 continues with the full decomposition. The optimizations and the resulting gate count are shown in Section 2.5. The chapter ends with the conclusion in Section 2.6.

2.2. NOTATION AND GATE DEFINITIONS

This section introduces the mathematical notation and gate definitions used in this chapter.

2.2.1. MATHEMATICAL OPERATIONS

The conjugate transpose of a matrix is represented with † (i.e. the conjugate transpose of matrix U is $U^\dagger$). Reversible quantum operations (gates) can be fully represented as unitary matrices, for which $U^\dagger = U^{-1}, UU^\dagger = I$, where I is the identity matrix.

The Kronecker product of two matrices is written as $\otimes$. The Kronecker product of $(n \times m)$ matrix A and $(p \times q)$ matrix B is the $(pm \times qn)$ block matrix:

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}$$

The Kronecker sum of two matrices is written as $\oplus$. The Kronecker sum of $(n \times m)$ matrix A and $(p \times q)$ matrix B is the $((m+p) \times (n+q))$ block matrix:

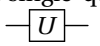
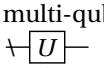
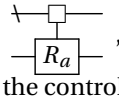
$$A \oplus B = \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}$$

where the zeros are zero matrices.

2.2.2. GENERIC GATES

The elementary quantum operations used in this chapter are part of the well-established and widely used set presented in [19].

The following generic gates are used in the decomposition and their circuit representation are listed below.

- Generic single-qubit unitary gate:
  $U(2) =$ 
- Generic multi-qubit unitary gate:
  $U(n) =$  where the backslash is used to show that the wire carries an arbitrary number of qubits.
- Controlled arbitrary (multi-qubit) gates:
   $= \begin{bmatrix} I & 0 \\ 0 & U \end{bmatrix}$, gate U is only applied if the control qubit is in state $|1\rangle$.
- Quantum multiplexor:  $= U_1 \oplus U_2 = \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix}$, gate U_1 is applied if the control qubit is in state $|0\rangle$, gate U_2 is applied if the control qubit is in state $|1\rangle$.
- Uniformly controlled rotation gate:
   , a different rotation around axis a is applied depending on the state of the control qubits.

2.3. DECOMPOSING UNIFORMLY CONTROLLED ROTATIONS

This section shows the decomposition for one of the main building blocks resulting from our method; the uniformly controlled rotation gates. These gates will be decomposed using the method from [140].

The uniformly controlled rotation gates that are used in our decomposition method are always uniformly controlled R_z gates applied to the first qubit. The matrix representation of such a gate follows from the general matrix representation of a uniformly controlled R_z gate with k controlling qubits, and is $(D \oplus D^\dagger)$, where D is a $(2^k \times 2^k)$ diagonal matrix.

This gate can be implemented by an alternating sequence consisting of 2^k CNOTs and 2^k single qubit rotation gates applied to the target qubit. The CNOT controls are determined using a sequence based on the binary reflected Gray code [71]. The 2^k rotation gates in the circuit each apply a rotation by some angle θ_j to the target qubit, which can be calculated in such a way that the complete circuit is equivalent to $(D \oplus D^\dagger)$ [140].

The structure of the decomposition of a uniformly controlled R_z gate with three control qubits is shown in Figure 2.1.

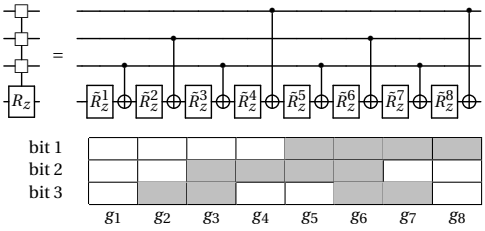


Figure 2.1: Decomposition of a uniformly controlled R_z gate with (k=3) control qubits with the three-bit Gray code that is used to find the control nodes of the CNOTs. In this figure $\tilde{R}_z^j$ is used to mean $R_z(\theta_j)$, where $j = 1, \cdots, 2^k$

2.4. FULL DECOMPOSITION

In this section, we first introduce the basis of our decomposition: the block-ZXZ decomposition [50]. Then we show how to decompose the circuit into elementary gates. This decomposition method results in the same number of CNOT gates as the unoptimized quantum Shannon decomposition [50].

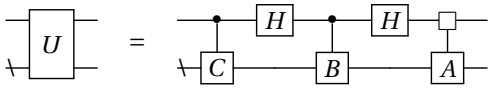
2.4.1. BLOCK-ZXZ DECOMPOSITION

The proposed decomposition is based on the block-ZXZ decomposition presented in [50], which shows how the method presented in [62] can be used to decompose a general unitary gate into the following structure:

$$U = \frac{1}{2}\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}\begin{bmatrix} I+B & I-B \\ I-B & I+B \end{bmatrix}\begin{bmatrix} I & 0 \\ 0 & C \end{bmatrix} \tag{2.1}$$

$$= \frac{1}{2}\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}(H \otimes I)\begin{bmatrix} I & 0 \\ 0 & B \end{bmatrix}(H \otimes I)\begin{bmatrix} I & 0 \\ 0 & C \end{bmatrix} \tag{2.2}$$

This can be represented as the following quantum circuit:



To construct this circuit, we need to solve Equation (2.1), which requires that [62]:

$$U\begin{bmatrix} I \\ C^\dagger \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \tag{2.3}$$

To find matrices A_1, A_2 and C, we first divide the starting matrix U into four equal blocks. We call the upper left block X, the upper right block Y and the lower two blocks U_{21} and U_{22}. This makes $U = \begin{bmatrix} X & Y \\ U_{21} & U_{22} \end{bmatrix}$ and then use singular value decomposition to decompose X and Y.

For X, with singular value decomposition we get $X = V_X \Sigma W_X^\dagger$ with unitary matrices $V_X, W_X \in U$ and Σ is a diagonal matrix with non-negative real numbers on the diagonal. We define $S_X = V_X \Sigma V_X^\dagger$, a positive semi-definite matrix and unitary matrix $U_X = V_X W_X^\dagger$. Then we have the polar decomposition of $X = S_X U_X$. The same method can be used to find S_Y and U_Y so that $Y = S_Y U_Y$.

Then we can write

$$U = \begin{bmatrix} S_X U_X & S_Y U_Y \\ U_{21} & U_{22} \end{bmatrix} \tag{2.4}$$

and define $C^\dagger = iU_Y^\dagger U_X$ so that Equation (2.3) becomes

$$U\begin{bmatrix} I \\ iU_Y^\dagger U_X \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \tag{2.5}$$

We can find $A_1 = (S_X + iS_Y)U_X$ and $A_2 = U_{21} + U_{22}(iU_Y^\dagger U_X)$. Finally, we rewrite Equation (2.1) and solve for the upper left corner to get $B = 2A_1^\dagger X - I$.

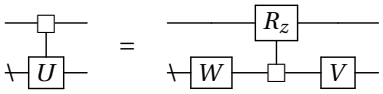
2.4.2. DEMULTIPLEXING

A gate $U = U_1 \oplus U_2$ can be decomposed into unitary matrices V and W and a unitary diagonal matrix D so that $U = (I \otimes V)(D \oplus D^\dagger)(I \otimes W)$ using the method described in theorem 12 of [173]:

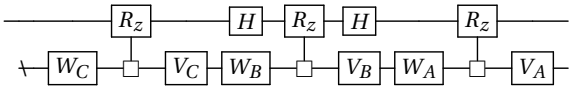
$$\begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} = \begin{bmatrix} V & 0 \\ 0 & V \end{bmatrix}\begin{bmatrix} D & 0 \\ 0 & D^\dagger \end{bmatrix}\begin{bmatrix} W & 0 \\ 0 & W \end{bmatrix} \tag{2.6}$$

To find the values for V, D and W, we first use diagonalization of $U_1U_2^\dagger$ to get $U_1U_2^\dagger = VD^2V^\dagger$, where V is a square matrix with columns representing the eigenvalues of $U_1U_2^\dagger$ and D a diagonal matrix whose diagonal entries are the corresponding eigenvalues. Then we can find W as $W = DV^\dagger U_2$. The matrix $D \oplus D^\dagger$ corresponds to a multiplexed R_z gate acting on the most significant qubit in the circuit.

In a quantum circuit, demultiplexing looks like this:



We can use this method to demultiplex gates A, B and C from the circuit in Section 2.4.1, which gives the following circuit:



It is clear from the circuit that gate V_C can be merged with W_B, and that V_B can be merged with W_A. This means we now have a circuit decomposition of an initial n-qubit gate into four (n-1)-qubit gates, three uniformly controlled R_z gates and two Hadamard gates. The uniformly controlled R_z gates can be decomposed as in Section 2.3. The decomposition is applied recursively to each (n-1)-qubit gate until only one-qubit gates are left, which can be decomposed using ZYZ-decomposition [19].

This leads to a total CNOT count that is the same as the unoptimized quantum Shannon decomposition [173]: $(3/4)\cdot 4^n - (3/2)\cdot 2^n$.

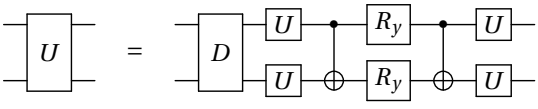
2.5. OPTIMIZATION

Because the circuit resulting from the block-ZXZ decomposition is very similar to that of the quantum Shannon decomposition [139, 173], it can be optimized using the same methods. But where QSD can merge one CNOT gate from the central R_y gate, we can merge two CNOT gates into the central multiplexor. This results in a total CNOT count of $\frac{22}{48}4^n - \frac{3}{2}2^n + \frac{5}{3} = \frac{11}{24}4^n - \frac{3}{2}2^n + \frac{5}{3}$ CNOT gates for decomposing an n-qubit unitary gate.

2.5.1. DECOMPOSITION OF TWO-QUBIT OPERATORS

The decomposition can be applied recursively until the biggest blocks are the generic 2-qubit unitary gates. These can be decomposed using the optimal 3-CNOT circuit, which can be done using one of several methods [174, 176, 182]. This reduces the CNOT count to $(9/16)\cdot 4^n - (3/2)\cdot 2^n$ [173].

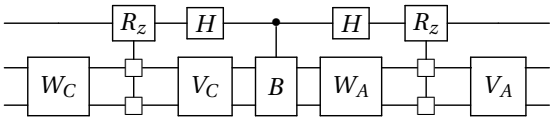
The CNOT count can be further reduced using the technique first described in [175]. The right-most two-qubit gate can be decomposed up to the diagonal into the following circuit, which requires only two qubits [174]:



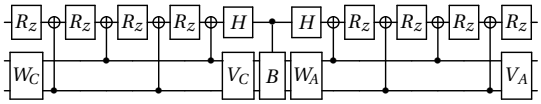
The diagonal matrix can be migrated through the circuit and merged with the next two-qubit gate, which can then be decomposed and its diagonal joined with the next, until only one two-qubit gate is left. This reduces the CNOT count by $4^{n-2}-1$ gates to $(8/16)\cdot 4^n-(3/2)\cdot 2^n-1$.

2.5.2. MERGING TWO CNOT GATES INTO THE CENTRAL MULTIPLEXOR

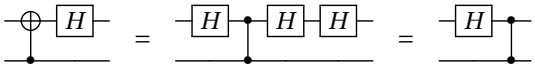
After the block-ZXZ decomposition, we first decompose only the left and right multiplexors (A and C). We now have a circuit with two uniformly controlled R_z gates. Using the decomposition of a three-qubit unitary as an example, the circuit now looks like this:



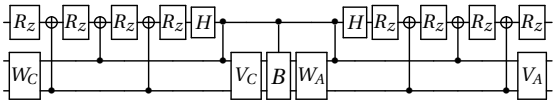
When decomposing the uniformly controlled R_z gates (see Section 2.4.2), we can modify one of the decompositions so that both of the Hadamard gates are next to a CNOT:



The Hadamard gates can be moved to the other side of two CNOTs, making them into CZ gates:



This makes the circuit:

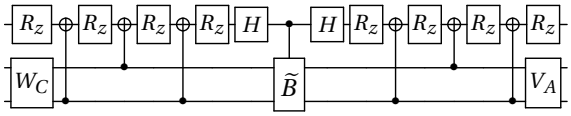


The two CZ gates can be merged into the middle controlled gate (B) together with the two (n-1)-qubit gates V_C and W_A, similar to the optimization introduced in [139].

The new central gate $\widetilde{B}$ can be calculated as:

$$\begin{aligned}\widetilde{B} &= (CZ\otimes I)\begin{bmatrix} W_A & 0 \\ 0 & W_A \end{bmatrix}\begin{bmatrix} I & 0 \\ 0 & B \end{bmatrix}\begin{bmatrix} V_C & 0 \\ 0 & V_C \end{bmatrix}(CZ\otimes I) \\ &= \begin{bmatrix} W_A V_C & 0 \\ 0 & (Z\otimes I)W_A B V_C (Z\otimes I) \end{bmatrix} \end{aligned} \tag{2.7}$$

and decomposed as a regular multiplexor, using the method described in Section 2.4.2.



This saves two CNOTs for every step of the recursion, for a total savings of $2\cdot(4^{n-2}-1)/3$ CNOT gates when stopping the recursion at generic two-qubit gates.

2.5.3. GATE COUNT

Figure 2.2 shows the structure of the circuit after the decomposition of a generic three-qubit gate. For a three-qubit unitary, the decomposition results in four generic two-qubit gates. Three of these require two CNOTs to implement, while the last one requires three CNOTs. The left and right controlled R_z gates both need three CNOTs and the middle uniformly controlled R_z gate requires four CNOTs to decompose. That makes the total CNOT count for the decomposition of a three-qubit unitary: $3 \cdot 2 + 3 + 2 \cdot 3 + 4 = 19$ CNOT gates.

To find the number of CNOT gates required for implementing bigger operators, we start with the recursive relation below. An n-qubit unitary requires c_n CNOTs, which are at most:

$$c_n \leq 4 \cdot c_{n-1} - 3 + 3 \cdot 2^{n-1} - 2 = 4 \cdot c_{n-1} + 3 \cdot 2^{n-1} - 5$$

This breaks down as follows: at each level of the recursion, the CNOT count is the sum of the CNOTs required for the decompositions of the four smaller unitaries (c_{n-1}) and the CNOTs required by the three quantum multiplexors (2^{n-1}). Three of the smaller unitaries can be implemented using one CNOT less by applying the optimization presented in Section 2.5.1, and two of the multiplexors can be implemented using one less CNOT using the method in Section 2.5.2.

A two-qubit unitary operator can be decomposed using at most three CNOTs ($c_2 \leq 3$), the recursive relations for 3, 4 and 5 qubit unitary operators are given below.

$$\begin{aligned}
c_3 &\leq 4 \cdot c_2 + 3 \cdot 2^{3-1} - 5 \\
c_4 &\leq 4 \cdot c_3 + 3 \cdot 2^{4-1} - 5 \\
&\leq 4 \cdot 4 \cdot c_2 + 4 \cdot 3 \cdot 2^{4-2} - 4 \cdot 5 + 3 \cdot 2^{4-1} - 5 \\
&\leq 4^2 \cdot c_2 + 3 \cdot 2^{4-1}(4 \cdot 2^{-1} + 1) - 5 \cdot (4 + 1) \\
c_5 &\leq 4^3 \cdot c_2 + 3 \cdot 2^{5-1}(4^2 \cdot 2^{-2} + 4 \cdot 2^{-1} + 1) - 5 \cdot (4^2 + 4 + 1) \\
&\leq 4^3 \cdot c_2 + 3 \cdot 2^{5-1}(2^2 + 2^1 + 1) - 5 \cdot (4^2 + 4 + 1)
\end{aligned}$$

We can recognize the following structure [33]:

$$1 + x + x^2 + \cdots + x^n = \frac{x^{n+1} - 1}{x - 1}$$

We can use this to derive the following relation for the CNOT count for the decomposition of an n-qubit unitary gate:

$$\begin{aligned}
c_n &\leq 4^{n-2} \cdot c_2 + 3 \cdot 2^{n-1}\left(\frac{2^{(n-3)+1} - 1}{2 - 1}\right) - 5\left(\frac{4^{(n-3)+1} - 1}{4 - 1}\right) \\
c_n &\leq 4^{n-2} \cdot c_2 + 3 \cdot 2^{n-1}(2^{n-2} - 1) - \frac{5}{3}(4^{n-2} - 1) \\
c_n &\leq \left(4^{-2} \cdot c_2 + 3 \cdot 2^{-3} - \frac{5}{3} \cdot 4^{-2}\right) \cdot 4^n - 3 \cdot 2^{-1} \cdot 2^n + \frac{5}{3}
\end{aligned}$$

With $c_2 \leq 3$, we get the following CNOT count for the decomposition of an n-qubit unitary gate:

$$c_n \leq \left(\frac{3}{16} + \frac{3}{8} - \frac{5}{48}\right) \cdot 4^n - \frac{3}{2} \cdot 2^n + \frac{5}{3}$$
$$c_n \leq \frac{22}{48} \cdot 4^n - \frac{3}{2} \cdot 2^n + \frac{5}{3}$$

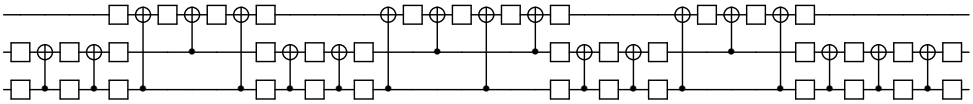


Figure 2.2: Decomposition of a three-qubit gate using 19 CNOTs and 37 single qubit gates.

2.6. CONCLUSION

In this chapter, we presented a novel quantum decomposition method that is able to produce circuits with a gate count that is lower than existing state-of-the-art quantum decomposition methods. We used the optimizations presented by [139] and [173], gate commutation and gate merging to optimize the block-ZXZ decomposition [50]. The decomposition follows the same structure as the well-known quantum Shannon decomposition, and has the same benefit of using recursion on generic quantum gates. This means that the decomposition can take advantage of the known optimal decompositions for two-qubit unitary gates, and other small-circuit optimizations, heuristic methods or optimal decompositions for three or more qubit gates when these become available.

Other than general unitary gates, the decomposition uses only single qubit gates and diagonal gates. This simplifies the structure and presents further opportunity for optimizations, such as accounting for specific hardware constraints like connectivity.

The circuit output of the decomposition can be compiled to any universal gateset. The resulting circuit will have the same overall structure with an equal number of two-qubit gates when the gateset includes a two-qubit gate that is equivalent to the CNOT gate up to single qubit gates. This is the case for, among others, the CZ gate (part of the native gateset of the IBM Heron) [19], the ECR gate (IBM Eagle) [96] and the XX gate (trapped ions) [78]. The compilation to a different type of two-qubit gate will add additional single qubit gates, but many of these can be merged into neighboring (generic) gates.

If these circuits are executed on a quantum execution platform which has a more permissive gateset, the diagonal gates can also be implemented with uniformly controlled Z-gates [141] instead of CNOTs. QR, QSD and Cartan decompositions have also been generalized to higher-dimensional quantum systems [94], which may offer practical advantage over two-level qubits [119]. If our decomposition is also generalizable to multi-level quantum systems, it may result in more optimal gate counts for these types of systems as well.

As can be seen in Table 2.1, our approach improves upon the previous record holder by $(4^{n-2}-1)/3$ CNOT gates to achieve the best-known CNOT count for any generic quantum gate of size three or more qubits.

3

Efficient Decomposition of Unitary Matrices in Quantum Circuit Compilers

In this chapter, we present our implementation of a unitary decomposition algorithm in the programming framework OpenQL. Unitary decomposition is a widely used method to map quantum algorithms to an arbitrary set of quantum gates. Efficient implementation of this decomposition allows for the translation of bigger unitary gates into elementary quantum operations, which is key to executing these algorithms on existing quantum computers.

Unitary decomposition can be used as an aggressive optimization method for the whole circuit, as well as to test part of an algorithm on a quantum accelerator. For the selection and implementation of the decomposition algorithm, perfect qubits are assumed. We base our decomposition technique on quantum Shannon decomposition, which generates $O(\frac{3}{4}4^n)$ controlled-not gates for an n-qubit input gate.

In addition, we implement optimizations to take advantage of the potential underlying structure in the input or intermediate matrices, as well as to minimize the execution time of the decomposition. Comparing our implementation to Qubiter and the UniversalQCompiler (UQC), we show that our implementation generates circuits that are much shorter than those of Qubiter and not much longer than the UQC. At the same time, it is also up to 10 times as fast as Qubiter and about 500 times as fast as the UQC.

This chapter is based on the following article[1]:

- Anna M. Krol, Aritra Sarkar, Imran Ashraf, Zaid Al-Ars, and Koen Bertels. "Efficient Decomposition of Unitary Matrices in Quantum Circuit Compilers". In: *Applied Sciences* 12.2 (2022). ISSN: 2076-3417. DOI: 10.3390/app12020759

CODE AVAILABILITY

OpenQL can be found at: https://github.com/QuTech-Delft/OpenQL and the specific version and branch that were used to implement unitary decomposition and the optimizations can be found here: https://github.com/anneriet/OpenQL.

[1] This article was completed before the work presented in Chapter 2, and therefore did not originally include the improved block-ZXZ based decomposition.

3.1. INTRODUCTION

Quantum computing is promising to provide the next phase of performance improvement for large-scale computing. To this end, many different algorithms have been developed in the theoretical domain, such as Shor's algorithm for prime factorization in polynomial time [178], or Grover's algorithm for finding a specific input corresponding to some output in $\sqrt{N}$ time [74].

Recent years have seen some great strides in the field of physical implementations of quantum computers as well. However, these still have some big limitations on the number of qubits, the error rates and the length of the circuits that can be executed on them. Although quantum computers with as many as 128 qubits already exist [6], error rates are of the order $10^{-2} - 10^{-3}$ per gate [188]. Therefore, executing a circuit on a physical quantum chip requires significant error correction, as well as mapping, scheduling and other such measures [103].

These algorithms are executed on simulators, which come with their own set of restrictions. Some simulators require the use of specific qubit topology and limit possible qubit states or the number of qubits, and all of them are bound by the classical resources of the system the simulation is run on. The main resource limit is the memory necessary to store the quantum circuit and the total qubit state, which is dependent on the length of the circuit, the number of qubits and the degree of superposition. These also influence the processing time necessary to simulate the full circuit, which is generally done by some form of matrix multiplications of the qubit state and each gate in the circuit.

Unitary decomposition is the process of translating an arbitrary *unitary* gate [2] into a specific (universal) set of single and two-qubit gates. Unitary decomposition is necessary because it is not otherwise possible to execute an arbitrary quantum gate on either a simulator or quantum accelerator. This makes it a required feature for algorithms that use any type of gate that is not supported by the target platform or just produce an arbitrary unitary gate that will need to be translated. In this chapter, only exact decomposition algorithms will be considered for application on gate-based quantum computing.

This chapter proposes a highly-efficient method to implement unitary decomposition for quantum algorithms using Quantum Shannon Decomposition. The chapter shows that our approach is up to 10× more efficient in terms of the number of gates generated for a given unitary matrix size and requires up to 100 times less wall-clock execution time than other implementations. The contributions of this chapter are as follows:

- Implementation of Quantum Shannon Decomposition for the unitary decomposition of quantum algorithms;
- Decomposition optimizations that take advantage of the underlying matrix structure;
- The integration and evaluation of our method in the OpenQL quantum programming framework;

[2] a unitary matrix U is a square, complex matrix, of which the inverse (U^{-1}) and the conjugate transpose ($U^{\dagger}$) are the same; i.e., $U^{\dagger} = U^{-1}$ and $UU^{\dagger} = I$ [5]

- The optimization of the implementation of a quantum genome analysis use-case using our method.

This chapter is structured as follows: In Section 3.2, applications for unitary decomposition are discussed and some background is given on qubits, gate-based computation and the special qubit gates that are used this chapter. The specific decomposition method for multi-controlled gates is given in Section 3.3. In Section 3.4, several decomposition algorithms are compared based on their resulting CNOT-count. The implementation of the selected algorithm, Quantum Shannon Decomposition, is outlined in Section 3.5. Experimental results are shown in Section 3.6 and compared to other implementations in Section 3.7. Finally, the conclusion and future work can be found in Section 3.8.

3.2. BACKGROUND

In this section, we first discuss the motivation for unitary decomposition in Section 3.2.1. In Section 3.2.2, we will give the notation for the quantum and mathematical constructs that are used in this chapter.

3.2.1. MOTIVATION FOR UNITARY DECOMPOSITION

Unitary decomposition is useful in several contexts. The first is the broad class of algorithms that generate arbitrary unitary gates that need to be translated into a quantum circuit, but it is also used to enable the more modular design of quantum algorithms or as an aggressive optimization method.

We will use two quantum algorithms that we have developed in the context of genome sequencing as an example of a possible application for unitary decomposition. With genome sequencing, a genome sequence is first read as many short pieces which then need to be combined to get the full DNA sequence. This is currently done using many different algorithms, which are executed using (classical) high-performance computing systems [85].

For genome sequencing using quantum accelerators, the DNA sequences can be stored in superposition. The two algorithms that will be discussed both use a unitary matrix in the process of finding the position of a short read (sequence of a small piece of DNA) on a reference genome. That matrix needs to be decomposed before the algorithm can be run on a quantum accelerator or simulator [167].

The first quantum genome sequencing algorithm we will use is Quantum Indexed Bidirectional Associative Memory (QiBAM) [167]. QiBAM uses a unitary oracle $U(2^n) = I(2^n) - 2|b_p\rangle\langle b_p|$ assembled from a binomial distribution as $|b_p^x\rangle = \sqrt{\gamma^{h(p,x)}(1-\gamma)^{n-h(p,x)}}$. Here, γ is a factor that influences the width of the distribution, $h(p,x)$ is the Hamming distance between the query pattern p and all memory states x, and n is the number of qubits required to store the memory states. n is also the size of the vector and resulting matrix.

$$\hat{S}^{S_p p} = CR_y(2sin^{-1}(-1/\sqrt{p}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \sqrt{\frac{p-1}{p}} & \frac{-s}{\sqrt{p}} \\ 0 & 0 & \frac{s}{\sqrt{p}} & \sqrt{\frac{p-1}{p}} \end{bmatrix} \tag{3.1}$$

The second genome sequencing algorithm is Quantum Associative Memory (QAM). This uses a Hadamard-like transformation to store the patterns, assembled using Equation (3.1) [197].

In order to apply either gate from these two algorithms to qubits, they need to be translated into some combination of (elementary) quantum gates that can be executed on a quantum accelerator, and the same is true for other such algorithms.

Besides that, unitary decomposition also facilitates short-cuts in the design of new algorithms. With unitary decomposition, a developer can keep part of an algorithm as a unitary gate/matrix while working on some other part and test this. Otherwise, the algorithm can only be executed in full when all of it is made out of known quantum gates. Unitary decomposition allows the full algorithm to be tested and checked much earlier in the development process on the target quantum chip or simulator.

Furthermore, unitary decomposition can be used as an aggressive optimization method, because the maximum number of gates resulting from a decomposition can be calculated easily beforehand. The maximum length of the circuit resulting from the decomposition is only dependent on the number of qubits affected by the gate. For circuits longer than this maximum, and thus consisting of more gates, the assembly of all gates into a unitary matrix and then decomposing that matrix will always result in a shorter circuit.

Someone programming in OpenQL might, for example, specify a circuit with three qubits with 180 gates—this might be because of application semantics, code-readability or because they did not consider the optimal way to program their quantum algorithm. The total of 180 gates is more than the number of gates that would result from decomposing an arbitrary three-qubit gate. Thus, if the circuit is combined into a single unitary matrix and then that matrix is decomposed using Shannon Decomposition, for example, then the length of the circuit will be reduced from 180 gates to only 120 (84 rotation gates and 36 CNOT gates).

Something to consider, however, is that the circuit resulting from the decomposition of a unitary matrix is longer than the theoretical minimum, and even the theoretical minimum number of gates for a general n-qubit unitary gate becomes quite large very quickly, since it scales with 4^{n-1} in the leading term. Thus, in most cases, a hand-optimized and application-specific circuit will be shorter than the one resulting from universal unitary decomposition. However, these hand-optimized circuits are labor-intensive and require a significant amount of time to develop, while unitary decomposition can be done automatically.

3.2.2. NOTATION

In this section, the notations are given for qubits, quantum gates, unitary matrices, the universal set of gates that are used, quantum multiplexers and multi-controlled gates.

QUBIT NOTATION

A qubit state is represented in bracket notation as

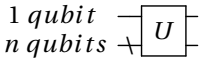
$$|\phi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{3.2}$$

Besides the $|\_\rangle$ notation, quantum states can also be represented as complex vectors: $\alpha|0\rangle+\beta|1\rangle = \begin{bmatrix}\alpha & \beta\end{bmatrix}^T$. This is especially useful for the combined state of multiple qubits, where the first row of the vector corresponds to the binary number "0" in as many bits as there are qubits. The second row corresponds to the number "1", etc. As an example, for a three-qubit state, the first row corresponds to $|000\rangle$, and the second to $|001\rangle$. This continues to the final row, which is $|111\rangle$. The state vector has 2^n rows for the state of n qubits.

QUANTUM GATES

Qubits are manipulated using gates, which are matrices that operate on the qubit state vector. To calculate the effect of gates on the combined qubit state, the state vector is multiplied by the matrix representations of the gates in reverse order.

In the circuit notation, each line going into or out of a gate represents one qubit. To represent n-qubit gates—gates that affect an unspecified number of qubits—a line with a backslash through it is used.

$(n+1)$-qubit gate = 

UNITARY MATRICES

A reversible quantum gate acting on n perfect qubits can be fully described as unitary matrix [52]. The set of unitary matrices of size 2^n by 2^n is written as $U(2^n)$ and has the following properties [5]:

- $U^\dagger = U^{-1}$;
- U is diagonalizable;
- $U(2^n)$ has a set of 2^n orthogonal eigenvectors;
- For a 2×2 unitary matrix $U = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$, $\sqrt{A^2+B^2} = 1$.

The Special Unitary group, SU, is a subgroup of unitary matrices where

- $|det(U)| = 1$ for U in SU [170]

When a measurement is performed, the global phase (Φ) of the qubits does not influence the measurement probabilities. This means that all quantum gate operations can be represented by a matrix in $SU(2^n)$ [32]. These properties are used to decompose the matrix, using one of the algorithms described in Section 3.4.

UNIVERSAL SET OF GATES

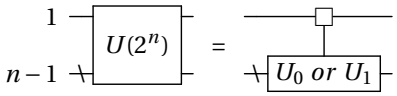
In order to decompose all possible unitary matrices into quantum gates, a universal gate set is selected as CNOT, the $R_z(\theta)$ and $R_y(\theta)$ gates. These three were selected because they are used in most decomposition methods.

QUANTUM MULTIPLEXERS

Besides these conventional gates, there are several gates used in this chapter as intermediate results for the various decomposition algorithms. The first is the quantum multiplexer, which corresponds to a unitary matrix corresponding to the following structure Equation (3.3).

$$U(2^n) = \begin{bmatrix} U_0(2^{n-1}) & 0 \\ 0 & U_1(2^{n-1}) \end{bmatrix} \tag{3.3}$$

where $U(2^n)$ denotes a unitary gate over n qubits, which is a unitary matrix of 2^n rows and 2^n columns. $U_0(2^{n-1})$ and $U_1(2^{n-1})$ are both $(n-1)$-qubit gates. The rest of the matrix of U is zero. The gate is uniformly controlled, which means that when the control is 0, the upper left (U_0) of the matrix affects the qubits. However, when the control is 1, the lower right gate (U_1) is applied. The circuit for this is shown in Circ. 3.1.



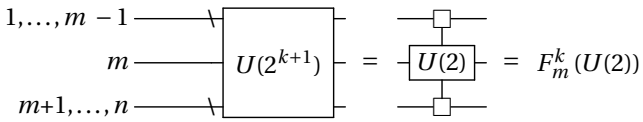
Circuit 3.1: A quantum multiplexer

The first line is the controlling qubit, and the lower line is the rest $(n-1)$ of the qubits. The box with the line to the lower gate means that it is uniformly controlled.

MULTI-CONTROLLED (ROTATION) GATES

Another common intermediate gate is the multi-controlled (rotation) gate. This is a one-qubit gate with k control bits. Rather than just applying a gate when all control bits are zero, the applied operation to the target qubit can be different for each of the 2^k possible classical values of the control qubits.

This is written as $F_m^k(U(2))$, which is a fully or multi-controlled $U(2)$ gate with k control qubits, with the target qubit at position m. The circuit representation of this gate is shown in Circ. 3.2. To indicate that an operation is applied for either state of the control bits, a square control box is used.



Circuit 3.2: A multi-controlled $U(2)$ gate.

These multi-controlled gates correspond to a (block) diagonal unitary matrix, which is why they show up frequently in decomposition schemes. This is shown in Equation (3.4).

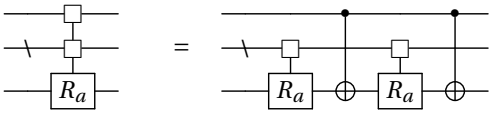
$$F_m^k(U(2)) = diag_j\left(U(2)_j\right) = \begin{bmatrix} U(2)_0 & & \\ & \ddots & \\ & & U(2)_{2^k} \end{bmatrix} \tag{3.4}$$

A multi-controlled rotation gate around axis a corresponds to the matrix shown in Equation (3.5). This can be any axis, but in this chapter, the multi-controlled R_y and R_z axes are mainly used.

$$F_m^k(R_a) = diag_j\left(R_a(\theta_j)\right) = \begin{bmatrix} R_a(\theta_0) & & \\ & \ddots & \\ & & R_a(\theta_{2^k}) \end{bmatrix} \tag{3.5}$$

3.3. DECOMPOSING MULTI-CONTROLLED ROTATION GATES

The multi-controlled rotation gates from Section 3.2.2 can be decomposed into a combination of CNOTs and regular rotation gates. This can be done using the method from [140], which results in 2^k CNOTs gates and 2^k 1-qubit rotation gates for a controlled rotation gate with k control bits. To move from an $F_k^m(R_a)$-gate to an $F_{k-1}^m(R_a)$-gate, a circuit such as Circ. 3.3 can be used.

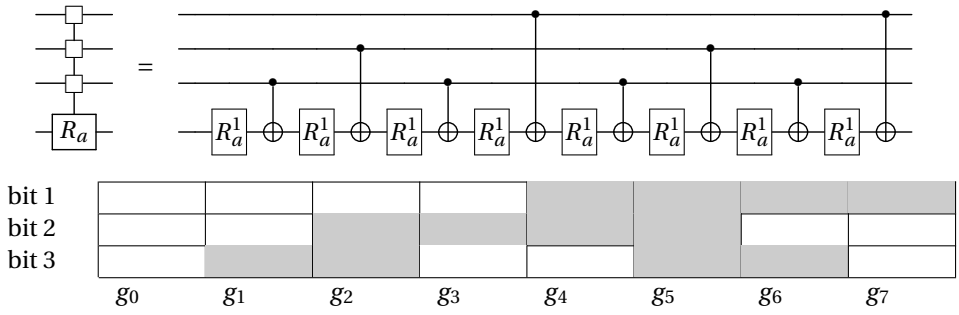


Circuit 3.3: Partial decomposition of an $F_k^m(R_a)$-gate.

This can be extended until only CNOT gates and one-qubit rotation gates are left, which leads to an example decomposition of a rotation gate with three control bits as shown in Circ. 3.4.

To directly calculate which qubit is the control bit for each CNOT, this can be determined using Gray code. This is shown in the table below the circuit. The number of the bit that is changed in the Gray code is the number of the qubit that will be the control bit.

For each control bit of the multi-controlled gate, a one-qubit rotation gate and a single CNOT is used, so the total decomposition of an F_m^k-gate requires 2^k rotation gates and CNOTs [140]. This is the least-known number of gates for decomposing such a matrix and is therefore used in almost all decomposition methods for (block) diagonal matrices of this form.



Circuit 3.4: Decomposition of an $F_4^3(R_a)$-gate.

3.4. COMPARISON OF DIFFERENT DECOMPOSITION METHODS

In this section, first, the selection criteria for the various decomposition methods is outlined in Section 3.4.1. Then, the theoretical lower bounds for the number of gates resulting from decomposition are given in Section 3.4.2, with implementations for a one and two-qubit gate in Sections 3.4.3 and 3.4.4. This is followed by an examination of various general decomposition methods from the literature in Sections 3.4.5–3.4.8 and finally the selection in Section 3.4.9.

3.4.1. SELECTION CRITERIA

Quantum computers are currently limited by the error rates and decoherence of qubits [188], and the longer the circuit, the higher the chance of errors will become. Therefore, the selection is based on circuit length, although the decomposition algorithm is only tested with perfect qubits on a simulator for now. In accordance with the motivations laid out in Section 3.2.1, only exact decomposition algorithms are considered.

For all decomposition methods, the number of gates resulting from the decomposition is only dependent on the number of qubits affected by the unitary gate. Thus, for generic n-qubit unitary gates, the resulting circuit length can be calculated from the size of the input matrix.

To measure the length of the resulting circuit, the number of CNOT gates will be used. There are several reasons for that. The first is that not all papers distinguish between generic one-qubit gates and rotation gates. The decomposition of a generic one-qubit gate takes three rotation gates (see Section 3.4.3), so the comparison might be a factor of three off if one-qubit gates are used to judge circuit length. The CNOT gate is used as the result for all decomposition methods and always has the same definition. This makes it a good metric for the total circuit length.

Secondly, each CNOT can generate entangled states between qubits [137], and for execution of the circuit on (near-term) quantum devices, each CNOT between non-neighboring qubits might introduce additional mapping operations [103]. Thus, to reduce mapping in the future, a circuit with as few CNOTs as possible is desired.

Thirdly, the error-rates for two-qubit gates are currently considerably higher than for one-qubit gates [188]. Thus, the chance that an error occurs in a circuit becomes much

bigger with more CNOTs. Thus, to make the decomposition feasible for near-term quantum applications, it is not only important to keep the circuit-length low but especially the CNOT count.

3.4.2. THEORETICAL LOWER BOUNDS

There is a theoretical lower bound for the number of CNOTs resulting from the decomposition of an n-qubit gate, and it is mathematically proven to be $\frac{1}{4}(4^n - 3n - 1)$ [176]. There are implementations that reach this number for one and two-qubit gates [176], as outlined in the next sections. This lower limit is included in the comparison, because it is useful to keep in mind what is and is not possible in terms of algorithms for unitary decomposition.

3.4.3. ZYZ DECOMPOSITION

For a one-qubit gate, no CNOT gates are necessary, and if rotation gates around any axis are possible, only one such gate is needed to apply any one-qubit operation. However, when using standard elementary gates, such as rotations around the Pauli X, Y or Z-axis, the decomposition of an arbitrary one-qubit gate results in three rotation gates using ZYZ decomposition [176].

One way to do this is with two rotation-z gates and one rotation-y gate. For this decomposition, the angles $\Phi, \alpha, \beta, \gamma$ can be found so that the following equation is satisfied:

$$U(2) = e^{-i\Phi}\begin{bmatrix} A & B \\ C & D \end{bmatrix} = e^{-i\Phi} R_z(\alpha) R_y(\beta) R_z(\gamma) \tag{3.6}$$

$$SU(2) = \begin{bmatrix} A & B \\ C & D \end{bmatrix} = R_z(\alpha) R_y(\beta) R_z(\gamma) \tag{3.7}$$

These angles can be calculated using the eigenvalues of the matrix and are used in the circuit shown in Figure 3.1. This is a universal decomposition for a one-qubit $SU(2)$ gate [176].

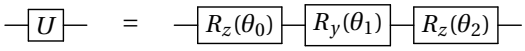


Figure 3.1: Minimal universal quantum circuit for a one-qubit gate [176].

3.4.4. MINIMAL DECOMPOSITION OF TWO-QUBIT GATES

From the theoretical lower bounds, we know that at least 2.25 CNOT gates are needed for a two-qubit gate. This rounds up to three CNOTs, and a circuit that achieves that number is shown in Figure 3.2 [176].

To obtain the values for the gates of this circuit, first angles α, β and δ are found as in the ZYZ decomposition (Section 3.4.3). These are used to make circuit v so that the following holds:

$$(a \oplus b)\, v\, (c \oplus d) = U(4) \tag{3.8}$$

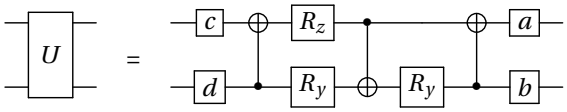


Figure 3.2: Minimal universal quantum circuit for a two-qubit gate using 18 elementary gates [176].

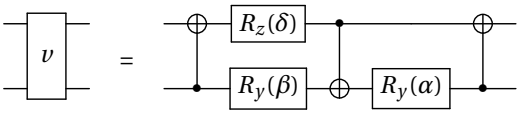


Figure 3.3: The circuit v used to construct a universal two-qubit gate [176].

The circuit v is shown in Figure 3.3.

Then, to get the one-qubit gates, first matrix $A \in SO(4)$ can be found so that $AUU^T A^\dagger$ is diagonal ($SO(n)$ is the special orthogonal group, which means that the inverse of a matrix Q is equal to its transpose: $Q^{-1} = Q^T$ and $det(Q) = 1$). Through more diagonalization, $B \in SO(4)$ can be found so $AUU^T A^T = Bvv^T B^T$ and matrix C as $C = v^\dagger B^T AU \in SO(4)$. This leads to $A^T BvC = U$, and because A, B and C are in the special orthogonal group, they can be implemented by two unitary gates. After combining A^T and B, the four gates can be found as [176]

$$A^T B = a \otimes b \tag{3.9}$$

$$C = c \otimes d \tag{3.10}$$

which gives the circuit in Figure 3.2. The four one-qubit gates can be implemented by three rotation gates each, through ZYZ decomposition, so that the total rotation count is $4 \cdot 3 + 3$ and the total CNOT count is just the ones for the circuit v, and thus three. This matches the theoretical lower bounds for an arbitrary two-qubit gate.

3.4.5. DECOMPOSITION WITH GIVENS ROTATIONS

In [194] a method of decomposition is described that uses the Givens rotation matrices to perform the QR factorization of a unitary matrix. Each Givens rotation nullifies the element on the ith column and jth row of a $U(2^n)$ matrix, as

$$^1G_{n,n-1}U = \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n-2,1} & u_{n-2,2} & \cdots & u_{n-2,n} \\ \tilde{u}_{n-1,1} & \tilde{u}_{n-1,2} & \cdots & \tilde{u}_{n-1,n} \\ 0 & \tilde{u}_{n,2} & \cdots & \tilde{u}_{n,n} \end{bmatrix} \tag{3.11}$$

The modified elements of U are indicated with a tilde, and the element on the lower left $u_{n,1}$ is nullified by the Givens rotation. Each Givens rotation matrix is equal to the identity matrix except for $c = cos(\theta)$ and $s = sin(\theta)$ for the elements at positions (i,i), (i,j),

(j,i) and (j,j), with θ the angle of the Givens rotation. These are to nullify elements until all elements below the diagonal are zero, at which point the following equality holds: [194]

$$\left(\prod_{i=1}^{2^n-1}\prod_{j=i+1}^{2^n} {}^{i}G_{j,j-1}\right)U = I \tag{3.12}$$

By reordering the base vectors according to Gray code (see Section 3.2.2), the cosine and sine elements will all be adjacent. This is convenient for quantum computation, because that means that each Givens rotation matrix can be implemented by a controlled one-qubit gate, C_i^k, with k control bits. For one specific combined state of the control qubits, the Γ gate is applied to qubit i, while for all other states, the target qubit is left unaffected. This means that

$$\prod_{i=1}^{2^n-1}\prod_{j=i+1}^{2^n} C_{\gamma(i)}^{n-1}(\Gamma_{(j,k)}^{\dagger}) = SU(2^n) \tag{3.13}$$

$${}^{i}\Gamma_{j,k} := \begin{bmatrix} {}^{i}g_{k,k} & {}^{i}g_{k,j} \\ {}^{i}g_{j,k} & {}^{i}g_{j,j} \end{bmatrix} \tag{3.14}$$

where $\gamma(i)$ denotes the ith number of the Gray code, and the gates ${}^{i}\Gamma_{j,k}$ are formed from the matrix for the Givens rotations. This results in the circuit shown in Figure 3.4 for the decomposition of a two-qubit gate.

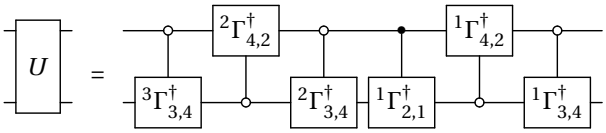


Figure 3.4: Decomposition into the Givens rotations [194].

The numbers of elementary gates and CNOTs were calculated using [19], which are the numbers included in the table. Generally, this decomposition requires approximately $8.4 \cdot 4^n$ controlled gates, which follows from a recursive relation of $g_n(k) = g_n^0(k) + g_{n-1}(k) + g_{n-1}(k-1)$ [194].

3.4.6. DECOMPOSITION THROUGH UNENTANGELING OF QUBITS

The first of the general decomposition methods uses consecutive unentangling of qubits, since one of the ways to specify an n-qubit gate is by its effect on the base vectors. This technique from [173] implements the correct behavior for each vector iteratively using a method similar to QR decomposition, which leaves previous assessed vectors preserved.

To get here, the qubit state vector is divided into 2^{n-1} vectors of each 2 elements, which are labeled $|\psi_j\rangle$ for $j = 0,\ldots,2^{n-1}-1$. For each of these vectors, Equation (3.15) is used to determine r_j, t_j, ϕ_j and θ_j.

$$|\psi\rangle = re^{it/2}\left[e^{-i\phi/2}cos\frac{\theta}{2}|0\rangle + e^{i\phi/2}sin\frac{\theta}{2}|1\rangle\right] \tag{3.15}$$

So that:

$$R_z(-\phi_j)R_y(-\theta_j)\,|\psi_j\rangle = r_j e^{it_j}\,|0\rangle \tag{3.16}$$

This corresponds to a circuit with a multi-controlled R_y and R_z, which is used to unentangle the last qubit. The new qubit vector is assembled as in Equation (3.17). The circuit to translate $|\phi\rangle$ into $|\phi'\rangle\,|0\rangle$ will be called E_k as in Equation (3.18).

$$|\psi'\rangle = \sum_{j=0}^{2^{n-1}-1} r_j e^{it_j}\,|j\rangle \tag{3.17}$$

$$F_n^{n-1}(R_y)F_n^{n-1}(R_z)\,|\psi\rangle = E_k\,|\psi\rangle = |\psi'\rangle\,|0\rangle \tag{3.18}$$

This circuit is implemented with the multi-controlled R_y and R_z gate, as is shown in Figure 3.5.

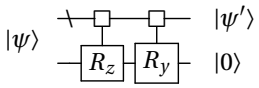


Figure 3.5: Unentangling a qubit state [173].

This method can be applied recursively to map an n-qubit state ϕ to a scalar multiple of a bit-string $|b\rangle$, which uses $2^{n+1}-2n$ CNOT gates.

Using this method to decompose unitary gates requires 2^{n-1} of these state preparation steps. At each step, a circuit C_j is found that maps a unitary gate U_j to a scalar multiple of $|j\rangle$ so that $U_{j+1} = C_j U_j$. The final product U_{2^n-1} will be a diagonal gate D, which can be implemented with a multi-controlled R_Z gate, so that $U(2^n) = C_0^\dagger C_1^\dagger \cdots C_{2^n-2}^\dagger D$.

Each circuit C_j needs $(2^{n+1}-n)$ CNOT gates, and with the final diagonal gate this leads to a total of $2\cdot 4^n - (2n+3)\cdot 2^n + 2n$ CNOT gates [173].

3.4.7. RECURSIVE COSINE SINE DECOMPOSITION

With the circuit presented in [191], an n-qubit gate is decomposed into multi-controlled rotation gates. Cosine sine decomposition (CSD) is applied recursively until all the matrices are diagonal.

With CSD, any even-dimensional unitary matrix U can be decomposed into real diagonal matrices C and S and smaller unitary matrices L_0, L_1, R_0 and R_1 as shown in Equation (3.19) [147].

$$U = \begin{bmatrix} U_{00} & U_{01} \\ U_{10} & U_{11} \end{bmatrix} = \begin{bmatrix} R_0 & 0 \\ 0 & R_1 \end{bmatrix} \begin{bmatrix} C & -S \\ S & C \end{bmatrix} \begin{bmatrix} L_0 & 0 \\ 0 & L_1 \end{bmatrix} \tag{3.19}$$

The left and right matrices are uniformly controlled gates; see Section 3.2.2. C and S are diagonal matrices with the cosines and sines of angles θ_j as the diagonal elements, respectively, between the subspaces, as shown in Equations (3.20) and (3.21).

$$C = \text{diag}(\cos(\theta_0), \ldots, \cos(\theta_n)) \tag{3.20}$$

$$S = \text{diag}(\sin(\theta_0), \ldots, \sin(\theta_n)) \tag{3.21}$$

where the values θ are ordered from large to small and are between $\frac{1}{2}\pi$ and 0.

The central matrices from each recursive step correspond to multi-controlled R_y gates which are decomposed as in Section 3.2.2. The other diagonal gates can be decomposed into a circuit consisting of $^1/_2 \cdot n \cdot 4^n - {}^1/_2 \cdot 2^n$ CNOTs and $^3/_2 \cdot 4^n - {}^1/_2 \cdot 2^n$ one-qubit rotation gates [51].

This is significantly improved upon in [139], which stops the recursion at uniformly controlled one-qubit gates.

Furthermore, this proves that any uniformly controlled two-qubit gate $(F_n^{n-1}(U(2)))$ can be decomposed into a specific sequence of $2^{n-1}-1$ CNOT gates, 2^{n-1} one-qubit gates and one total global phase gate expressed as Δ_n.

Furthermore, this proves that each multi-controlled two-qubit gate can be decomposed into a diagonal gate (Δ) and a Gray code sequence of CNOTs and one-qubit gates. The diagonal gates are folded into the central matrix from the CSD, so the total decomposition is

$$U = \Delta_n \tilde{F}_n^{n\text{-}1}(U(2)) \prod_{i=1}^{2^{n-1}\text{-}1} \tilde{F}_{n-\gamma(i)}^{n\text{-}1}(U(2)) \tilde{F}_n^{n\text{-}1}(U(2)) \tag{3.22}$$

Each $\tilde{F}_n^{n\text{-}1}(U(2))$ is decomposed with $2^{n-1}-1$ CNOTs, and the Δ_n gate is implemented with multi-controlled R_Z gates. This results in 2^n-2 CNOTs, which makes the total CNOT count $^1/_2 \cdot 4^n - {}^1/_2 \cdot 2^n - 2$. The resulting circuit is shown in Figure 3.6.

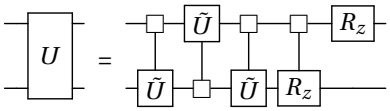


Figure 3.6: Recursive CSD decomposition [139].

3.4.8. QUANTUM SHANNON DECOMPOSITION

In [173], another way of using the CSD from Section 3.4.7, called Quantum Shannon Decomposition (QSD), is introduced. The decomposition of a two-qubit gate is shown in Figure 3.7.

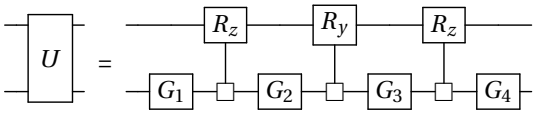


Figure 3.7: Quantum Shannon Decomposition [173].

The start of the decomposition is the same as in Section 3.4.7, but the L and R matrices are decomposed using eigenvalue decomposition. This is shown in Figure 3.8. The resulting matrices are unitary gates applied to one less qubit than the starting unitary. This leads to the circuit in Figure 3.7, where the D-matrix is implemented as a multi-controlled R_z gate.

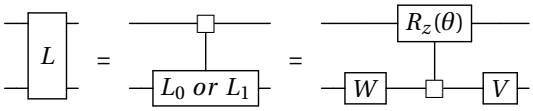


Figure 3.8: Decomposition of the L matrix in QSD [173].

Quantum Shannon Decomposition is applied recursively until the final one-qubit gates can be implemented with ZYZ decomposition. This means that only the multi-controlled rotation gates contribute to the number of CNOTs, each of which requires 2^{n-1} CNOT gates for a single step of the recursion of an n-qubit gate. This leads to a total CNOT count of $^3/_4 \cdot 4^n - {}^3/_2 \cdot 2^n$ for this decomposition method.

There are two optimizations that can be implemented on top of this implementation of Quantum Shannon Decomposition. The first is to stop the recursion at two-qubit gates and translate those as in Section 3.4.4. The second optimization is to implement the central multi-controlled R_z gate using CZ gates rather than CNOTs, of which one can be absorbed into the neighboring multiplexer. This results in one fewer CNOT gate at each level of the recursion. With these two implementations, the CNOT count comes to $^{23}/_{48} \cdot 4^n - {}^3/_2 \cdot 2^n + {}^4/_3$ [173] .

3.4.9. SELECTION OF THE ALGORITHM

For each decomposition method, the CNOT gate counts are compiled in Table 3.1 and plotted in Figure 3.9. As an indication, the number of CNOT gates resulting from the decomposition of a one to five-qubit unitary gate is given, along with the general formulas for the number of CNOT gates resulting from the decomposition of an n-qubit gate, if such a formula were available.

Table 3.1: CNOT counts for different implementations of unitary decomposition for one through five-qubit gates, as well as an n-qubit unitary gate. The chosen algorithm is shown in bold.

Number of Qubits	**1**	**2**	**3**	**4**	**5**	***n***	**Section**
Givens rotations [194]	0	4	64	536	4156	$\approx 8.4 \cdot 4^n$	Section 3.4.5
Iterative unentangling [173]	0	8	62	344	1642	$2 \cdot 4^n - (2n+3) \cdot 2^n + 2n$	Section 3.4.6
KG Cartan decomp. [129]	0	3	42	240	1128	$(21/16) \cdot 4^n - 3(n \cdot 2^{n-2} + 2^n)$	-
Recursive CSD [191]	0	14	92	504	2544	$(1/2) \cdot n \cdot 4^n - (1/2) \cdot 2^n$	Section 3.4.7
Recursive CSD (optimized) [139]	0	4	26	118	494	$(1/2) \cdot 4^n - (1/2) \cdot 2^n - 2$	Section 3.4.7
Block-ZXZ [50]	0	6	36	168	720	$(3/4) \cdot 4^n - (3/2) \cdot 2^n$	Chapter 2
QSD [173]	**0**	**6**	**36**	**168**	**720**	$(3/4) \cdot 4^n - (3/2) \cdot 2^n$	Section 3.4.8
QSD (optimized) [173]	0	3	20	100	444	$(23/48) \cdot 4^n - (3/2) \cdot 2^n + (4/3)$	Section 3.4.8
Beyond QSD	0	3	19	95	423	$(22/48) \cdot 4^n - (3/2) \cdot 2^n + (5/3)$	Chapter 2
Theoretical lower bounds [176]	0	3	14	61	252	$(1/2) \cdot (4^n - 3n - 1)$	Sections 3.4.2 to 3.4.4

Table 3.1, the decomposition introduced in Chapter 2 results in the fewest CNOT gates. But since that decomposition was not yet available when the work presented in this chapter was completed, the optimized QSD was the best decomposition available at the time. The choice was therefore made to implement QSD, although not the optimized version. The optimizations from [194] can be implemented without any modifications

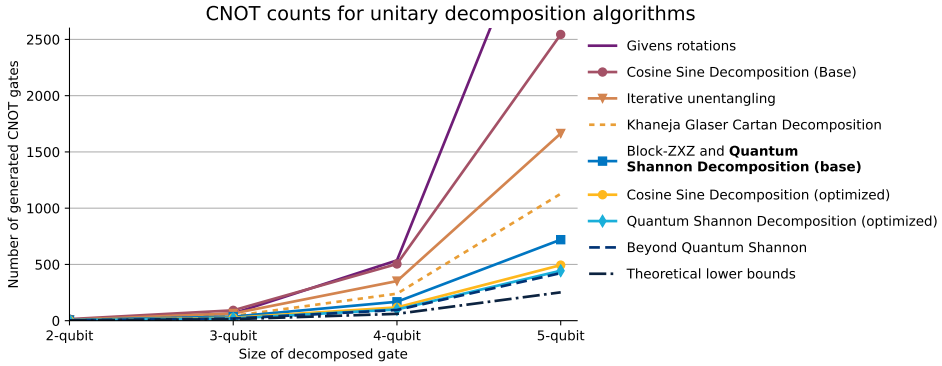


Figure 3.9: CNOT counts for different implementations of unitary decomposition for one through five-qubit gates. Scaling formulas and gate numbers can be found in Table 3.1.

to a base implementation of the algorithm.

Besides that, QSD has several other advantages. The recursion is performed at general n-qubit gates rather than multi-controlled one-qubit gates, which makes it relatively simple to implement. If algorithmic implementations for three-qubit, four-qubit or five-qubit or bigger general gates are found, they can be easily implemented. The same goes for other specific optimizations. In addition, because the mathematical decompositions are done separately for each step in the recursion, rather than all at once at the beginning, any underlying structure in the beginning or intermediate matrices can be taken advantage of immediately, therefore potentially skipping many computational steps as well as decreasing the size of the resulting circuit.

For these reasons, the choice was made to go with Quantum Shannon Decomposition for the implementation of unitary decomposition in OpenQL.

3.5. IMPLEMENTATION IN OPENQL

The implementation of the decomposition in OpenQL is split into two parts: the calculation of all of the rotation angles, and the generation of the circuit. This is done so that the implementation is independent from OpenQL. An example of unitary decomposition in OpenQL can be found in Code Listing 3.1.

For unitary decomposition in OpenQl, first a *Unitary* object is defined, which is then decomposed to calculate all the angles for all the rotation gates. The *Unitary* is then added to a *kernel* as any other gate. The *kernel* is added to a *program*, which is compiled with a *compiler*. The implementation is thus split between the *Unitary* class and the call to *kernel.gate()*.

Listing 3.1: Using unitary decomposition in OpenQL.

```
1 import os
2 from openql import openql as ql
3 import numpy as np
4 import sys
5
```

```
nqubits = int(sys.argv[1])

ql.set_option('output_dir', os.path.join(curdir, 'output'))
ql.set_option('log_level', 'LOG_ERROR');

platf = ql.Platform("starmon", os.path.join(curdir, 'config.json'))
program = ql.Program('example', platf, nqubits)
kernel = ql.Kernel("newKernel")

compiler = ql.Compiler("compiler1")

matrix = np.load("data/out_" + str(nqubits) + ".npy")
u1 = ql.Unitary("testname",matrix)
u1.decompose()
kernel.gate(u1, range(0,nqubits))
program.add_kernel(kernel)

compiler.compile(program)
```


3.5.1. THE UNITARY CLASS

The *Unitary* is instantiated with a string and an array. The content of this array is the unitary matrix, which is of size $2^n \times 2^n$ for an n-qubit gate. The complete Quantum Shannon Decomposition is computed only when "decompose()" is called, and the calculated angles for the resulting rotation gates are added to a list. This is done so that the *Unitary* can be used multiple times in a program without the recalculation of the whole decomposition.

However, before the decomposition is started, it is first checked whether the input matrix is unitary. If this is the case, all of the intermediate matrices will also be unitary [147], so this check is only necessary once. Furthermore, all of the Gray code matrices needed for the multi-controlled rotation gates are added to a lookup table so they do not need to be calculated anew at each decomposition step.

To make certain that the decomposition is correct, each single intermediate decomposition is checked. For each step, only three matrices need to be multiplied, and this saves any calculations that might be performed on an incorrect matrix. If any step of the decomposition is not correct, an exception is thrown and the decomposition is stopped.

The Eigen [1] library is used to perform singular value decomposition (SVD), eigenvalue decomposition and matrix multiplication. The recursion is centered on a main function, which takes as parameters a unitary matrix and the number of qubits. The latter is to keep track of the level of recursion.

Computation of the CSD is done using the method from [147], which uses SVD. The demultiplexing function uses Schur matrix decomposition for (sub)matrices smaller than $2^6 \times 2^6$ and eigenvalue decomposition for bigger matrices. This is done because Schur matrix decomposition is faster for small matrices [1].

The algorithm is recursive, and the demultiplexing step calls on the main function again for the decomposition of the smaller unitary matrices. If the matrices are of size 2×2, the rotation angles for the one-qubit rotation gates are calculated using ZYZ decomposition as in Section 3.4.3.

Because the *Unitary* does not have access to the qubit numbers of the circuit, only

the angles for the multi-controlled R_y and R_z are calculated at this point. This is done as in Section 3.2.2 by solving the following matrix equalities:

$$M^k \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_{2^k} \end{bmatrix} = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_{2^k} \end{bmatrix} \tag{3.23}$$

where M^k is a square matrix where all the entries are either "+1" or "−1", which are calculated using Gray code using Equation (3.24).

$$M^k_{ij} = (-1)^{b_(i-1)\cdot\gamma_(j-1)} \tag{3.24}$$

where the exponent is the bit-wise inner product of two binary vectors: b_i and γ_j. b_i is the integer i, and γ_j is the jth value of the Gray code.

For the multi-controlled R_y gate, the values of α_i are calculated by taking the arc sine of the diagonal entries of the S-matrix from the CSD.

$$\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_{2^k} \end{bmatrix} = 2 \cdot arcsin\left(S_{i,i}\right) \tag{3.25}$$

For the multi-controlled R_z gates, the values of α_i is calculated by taking the natural logarithm of the D-matrix from the demultiplexing.

$$\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_{2^k} \end{bmatrix} = -2 \cdot \sqrt{\text{-1}} \cdot ln\left(D_{(i,i)}\right) \tag{3.26}$$

All the angles for all rotation gates are added to a list, which is used to generate the correct gates when the *Unitary* is added to a circuit.

3.5.2. CIRCUIT ASSEMBLY

At the *kernel* level, when the (decomposed) *Unitary* object is added to the circuit, the gates and CNOTs are assembled and added to the circuit list. At this point, it is checked whether the *Unitary* is decomposed and if it is applied to the correct number of qubits. The first is checked from a flag that is set to "true" at the end of the decomposition. The latter is calculated from the size of the unitary matrix, which should be $2^n \times 2^n$ for an n-qubit gate.

Because the *kernel* only has the qubit numbers and the list of rotation angles, it does not have insight into whether any optimizations have happened. Therefore, the gates are added purely sequentially to the circuit, and each recursive call to the main function returns the total number of rotation angles used up until that point. If gates have been removed by an optimization, a specific angle is added to the circuit which signals how many gates have been removed, and these gates are skipped during circuit generation.

It is expected that the decomposition will take the most time to compute, as well as the most memory, since it contains the mathematical algorithms and matrix multiplications. Comparatively, using the calculated angles to make the circuit will not require much time or memory. Thus, adding the circuit sequentially is not expected to have much of an impact on the total resources required by the circuit, while it allows for a much more modular implementation of unitary decomposition.

3.5.3. COMPILATION OF THE OPENQL PROGRAM

After all gates have been added to the circuit, the *kernel* is added to a *program* which is compiled in OpenQL. From this point, the gates from the decomposition are handled in the same way as any manually added gates. Thus, the features and optimizations from the lower levels of the programming language can be fully used for the circuit [103]. Afterwards, the circuit is transformed into quantum assembly language and written to an output file as usual, or directly passed on to the simulator.

3.5.4. OPTIMIZATIONS

For the execution of the resulting circuit, it is important that it is as short as possible for the reasons mentioned in Section 3.4. To this end, the algorithm itself was selected to generate as few gates as possible. Combining and removing individual gates is performed in a later compile step by the OpenQL compiler [103], but more structural optimizations can be performed during the decomposition. For example, QAM, one of the algorithms from Section 3.2.1, generates a unitary matrix that has an internal structure that can be used to skip many steps in the recursion (see Section 3.2.1). The implemented optimizations take advantage of the matrix structure through the early detection of multiplexers and the detection of unaffected qubits.

DETECTION OF MULTIPLEXERS

Before the CSD is started, it is checked whether the upper right and lower left quarters of the matrix are already zero-matrices. If that is the case, the matrix already has the structure of a multiplexer and is directly passed to the demultiplexing step. This is signaled to the kernel by adding a specific gate angle to the list of rotation angles. This operation halves the number of resulting gates for this step of the decomposition.

UNAFFECTED QUBITS

If a decomposition step leaves a qubit unaffected, then it is not necessary to apply any gates to that qubit, and an n-qubit gate can be handled as an $(n-1)$-qubit gate. This reduces the resulting number of gates for this step by more than $^3/_4$. Thus, before the main decomposition is called, it is checked if the matrix is of the form $A \oplus I$ or $I \oplus A$. Each step of the QSD evaluates unitary gates on one less qubit, so any unaffected qubits become the first or last qubit at some point in the decomposition. If an unaffected qubit is detected, this is also signaled to the kernel. The unitary matrix of size $(n-1)$ is then assembled and passed back to the main function of the decomposition.

EXECUTION TIME OPTIMIZATIONS

There are also some optimizations to reduce the execution time and memory use of the decomposition.

One of the things done to reduce the total execution time and memory use is the fitting of ".noalias()" flags to all places where the product of multiple matrices is assigned to a different matrix. The Eigen library assumes aliasing for all such operations, and without this flag it evaluates the result of a matrix product into a temporary matrix that is then copied [1]. Another optimization is that all matrices are passed as references where possible to prevent any unnecessary copying of data.

The execution time and memory use of the decomposition after these and other optimizations can be found in Section 3.6. Circuit generation and its associated steps scale with approximately 2^{2n}, which is as expected since that implies a linear relation with the number of matrix elements and the length of the circuit. The decomposition itself scales as 2^{3n} [184].

3.6. EXECUTION TIME AND MEMORY ALLOCATION

The execution time of different parts of the decomposition is measured as the elapsed wall-clock time, with measurements in between function calls to determine the relative time consumption. The final execution times are shown in Figure 3.10. These tests were executed using a Dell Latitude 7400 with an 8th Generation Intel® Core™ i7-8665U Processor and 2× 4GiB DDR4 RAM.

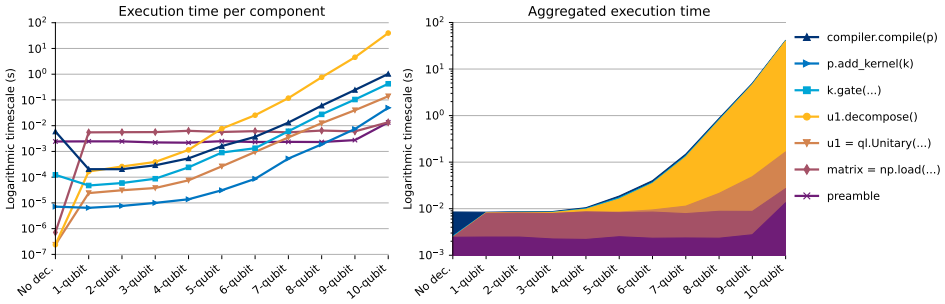


Figure 3.10: Execution time for the timed intervals, for the decomposition of different sizes of unitary matrices.

The program in Code Listing 3.1 has been used to determine the execution time and memory used by the decomposition. Unitary matrices of sizes $U(2^1)$ to $U(2^{10})$ were randomly generated first, using QiBAM as outlined in Section 3.2.1. The matrices were stored as binary files and loaded as required for the decompositions. The decomposition was repeated 1000 times for the smaller gates and 100 times for the decomposition of the 10-qubit gate in OpenQL, with varying numbers for the intermediate sizes. The execution time and memory use as reported in this chapter are the averages of these runs.

To measure execution time, the Python "time" package was used to determine the time difference between the start and various points of the program. The time for each part of the code, as well as the resulting aggregated execution time, can be found in Figure 3.10 and Table 3.2.

As expected, the decomposition itself took the most time—more than 10 times that

of any other part. This is because of the considerable mathematical decompositions and the number of matrix operations. One of the algorithms used in the decomposition is eigenvalue decomposition, which is an iterative algorithm that requires $O(6^n)$ operations for an $2^n \times 2^n$ matrix [23]. The data also show that the generation of the rotation gates and CNOTs does not contribute much to the total execution time of the algorithm, as expected. In addition, since the complete decomposition is calculated at design time, it does not influence the run-time of the final circuit when it is executed on a quantum accelerator.

Table 3.2: Total execution time at each line of Listing 3.1 for the decomposition of matrices of different sizes, in seconds.

Line	No dec.	1-qubit	2-qubit	3-qubit	4-qubit	5-qubit	6-qubit	7-qubit	8-qubit	9-qubit	10-qubit
Preamble	$2.43{\cdot}10^{-3}$	$2.46{\cdot}10^{-3}$	$2.45{\cdot}10^{-3}$	$2.24{\cdot}10^{-3}$	$2.18{\cdot}10^{-3}$	$2.50{\cdot}10^{-3}$	$2.32{\cdot}10^{-3}$	$2.35{\cdot}10^{-3}$	$2.32{\cdot}10^{-3}$	$2.75{\cdot}10^{-3}$	$1.34{\cdot}10^{-2}$
matrix = np.load(..)	$2.43{\cdot}10^{-3}$	$7.96{\cdot}10^{-3}$	$8.04{\cdot}10^{-3}$	$7.87{\cdot}10^{-3}$	$8.49{\cdot}10^{-3}$	$8.20{\cdot}10^{-3}$	$8.45{\cdot}10^{-3}$	$7.84{\cdot}10^{-3}$	$8.79{\cdot}10^{-3}$	$8.75{\cdot}10^{-3}$	$2.70{\cdot}10^{-2}$
u1 = ql. Unitary(..)	$2.43{\cdot}10^{-3}$	$7.99{\cdot}10^{-3}$	$8.07{\cdot}10^{-3}$	$7.91{\cdot}10^{-3}$	$8.57{\cdot}10^{-3}$	$8.46{\cdot}10^{-3}$	$9.40{\cdot}10^{-3}$	$1.13{\cdot}10^{-2}$	$2.13{\cdot}10^{-2}$	$4.86{\cdot}10^{-2}$	$1.66{\cdot}10^{-1}$
u1. decompose()	$2.43{\cdot}10^{-3}$	$8.15{\cdot}10^{-3}$	$8.33{\cdot}10^{-3}$	$8.30{\cdot}10^{-3}$	$9.71{\cdot}10^{-3}$	$1.61{\cdot}10^{-2}$	$3.49{\cdot}10^{-2}$	$1.29{\cdot}10^{-1}$	$7.82\cdot10^{-1}$	$4.60\cdot10^{0}$	$3.98\cdot10^{1}$
k.gate(..)	$2.56{\cdot}10^{-3}$	$8.20{\cdot}10^{-3}$	$8.39{\cdot}10^{-3}$	$8.39{\cdot}10^{-3}$	$9.95{\cdot}10^{-3}$	$1.70{\cdot}10^{-2}$	$3.63{\cdot}10^{-2}$	$1.35\cdot10^{-1}$	$8.09{\cdot}10^{-1}$	$4.70\cdot10^{0}$	$4.02\cdot10^{1}$
p.add_ kernel(k)	$2.56{\cdot}10^{-3}$	$8.21{\cdot}10^{-3}$	$8.39{\cdot}10^{-3}$	$8.40{\cdot}10^{-3}$	$9.97{\cdot}10^{-3}$	$1.71{\cdot}10^{-2}$	$3.63{\cdot}10^{-2}$	$1.36{\cdot}10^{-1}$	$8.11{\cdot}10^{-1}$	$4.71\cdot10^{0}$	$4.03\cdot10^{1}$
compiler. compile(p)	$8.49{\cdot}10^{-3}$	$8.41{\cdot}10^{-3}$	$8.60{\cdot}10^{-3}$	$8.69{\cdot}10^{-3}$	$1.05{\cdot}10^{-2}$	$1.87{\cdot}10^{-2}$	$4.00\cdot10^{-2}$	$1.49\cdot10^{-1}$	$8.71\cdot10^{-1}$	$4.95{\cdot}10^{0}$	$4.13{\cdot}10^{1}$

The same program has also been used to determine the memory allocation. This has been measured using the Python memory_profiler package. The results of this are shown in Table 3.3 and Figure 3.11. After an initial allocation of about 40 MiB, noteworthy additional allocation of memory occurs only when k.gate(...) is called. This means that the complete unitary decomposition requires much less memory than generating and storing the resulting circuit in OpenQL.

Table 3.3: Additional memory allocated at each line of Listing 3.1 for the decomposition of unitary matrices of different sizes, in MiB.

Line	1-qubit	2-qubit	3-qubit	4-qubit	5-qubit	6-qubit	7-qubit	8-qubit	9-qubit	10-qubit
Initial	43.078	43.117	42.973	43.172	43.102	42.914	42.906	43.180	43.063	43.082
matrix = np.load(...)	0	0	0	0	0	0	0	0.734	1.375	4.570
u1 = ql.Unitary(..)	0	0	0	0	0	0.766	1.855	3.258	12.160	48.141
u1.decompose()	0	0	0	0	0.820	0.867	1.945	5.750	12.156	46.184
k.gate(..)	0	0	0	1.230	0.660	1.711	6.441	27.582	120.65	483.65
p.add_kernel(k)	0	0	0	0	0	0	0.316	1.344	4.441	18.105
compiler.compile(p)	0	0	0	0	0.313	0.328	1.535	6.039	24.141	16.137

3.7. COMPARISON TO OTHER PROGRAMMING LANGUAGES

We compare our OpenQL implementation to Qubiter and UniversalQCompiler. These two are the only other quantum programming languages that, at the time of writing, also offer unitary decomposition.

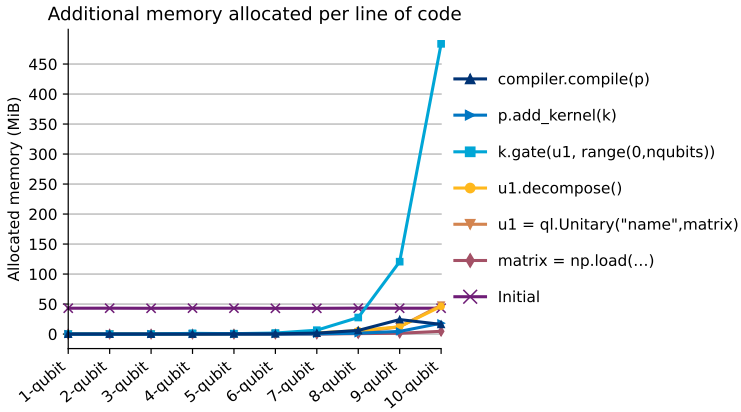


Figure 3.11: Additional memory allocated per line, for different sizes of unitary matrices

3.7.1. QUBITER

Qubiter [191] is a quantum compiler/programming language that aims to provide a set of tools for designing and simulating quantum circuits. As part of that, they offer unitary decomposition based on the recursive CSD from Section 3.4.7. Qubiter is written in Python and uses numpy for the mathematics, as well as the LAPACK cuncsd function for the CSD [51].

3.7.2. UNIVERSALQCOMPILER

UniversalQCompiler (UQC) is a software package written in the Mathematica language that can be used to decompose various quantum operations into CNOT gates and single qubit rotation gates. The resulting circuits can be displayed graphically or translated to OpenQASM, a quantum assembly language used by IBMQ, among others [93].

One of the types of decomposition they have implemented is unitary decomposition using QSD from [92]. This method produces $^{23}/_{48} \cdot 4^n - {}^3/_2 \cdot 2^n + {}^4/_3$ CNOTs, which is the same number as in [173].

3.7.3. COMPARISON RESULTS

We compare the execution time and the number of gates our implementation generates against Qubiter and UQC.

To obtain the total gate count, we use the number of lines in the output quantum assembly, which also includes rotation gates and not just CNOTs. The results for OpenQL, Qubiter and UQC are plotted in Figure 3.12. All of the implementations use an exact decomposition algorithm and therefore generate a constant number of gates for each size of the (non-sparse) input matrix.

It is clear that OpenQL always generates fewer gates than Qubiter, and almost all of the difference is in the number of CNOTs. This is because we use QSD in our implementation of unitary decomposition in OpenQL. For a 5-qubit gate, unitary decomposition with OpenQL generates a third of the CNOTs of Qubiter, and produces a total circuit that

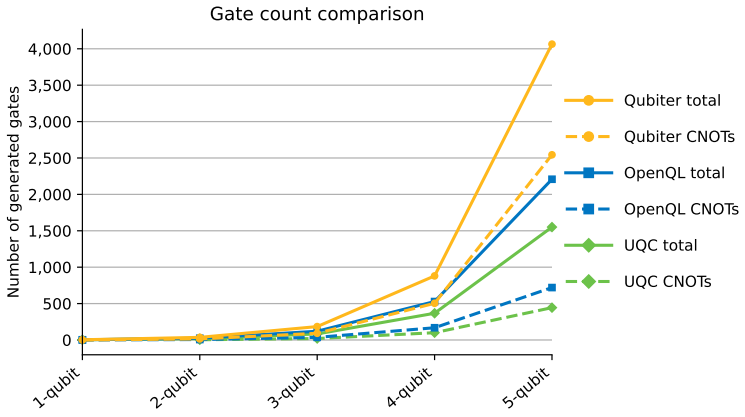


Figure 3.12: Number of generated CNOTs and total gates for OpenQL, UQC and Qubiter from the decomposition of different sizes of unitary matrices.

is almost twice as short.

When compared to UQC, our OpenQL implementation generates about 50% more of any type of gate. This difference is because UQC uses an optimal circuit at the two-qubit gate level, where OpenQL uses another iteration of QSD and then ZYZ-decomposition, which results in more gates.

The implementations are also compared on the time used to compute the unitary decompositions. The total wall-clock execution time for the decomposition and circuit generation of 2 to 10-qubit unitary gates can be found in Figure 3.13. The total execution time of all decompositions scales approximately linearly with the input matrix size (4^n for an n-qubit gate) for the decomposition of small matrices due to matrix loading and circuit generation operations and then becomes 8^n when the decomposition of bigger matrices begins to take more time than the other steps. This is around the decomposition of seven-qubit unitary gates.

As can be seen in the figure, OpenQL is considerably faster than Qubiter and UQC. When comparing the total execution times, it becomes clear that the OpenQL implementation takes more time per input matrix element (8^n) due to the use of CSD. Qubiter does not have that issue, but using unitary decomposition in OpenQL is about 10 to 100 times faster for the decomposition of 1 to 10-qubit unitary gates. This can most likely be attributed to the languages the compilers are programmed in and how well the implementation takes advantage of the programming language. Qubiter is written in Python and the UQC is written in Mathematica, both of which are considerably slower than C++, used for OpenQL [10].

In addition to being faster, unitary decomposition in OpenQL generates a much shorter circuit for all sizes of unitary matrices compared to Qubiter. For UQC, the tests were stopped at the decomposition of an eight-qubit unitary gate, which took approximately 450 s. Decomposing a nine-qubit gate was stopped after an hour, when it had still not produced results. As a result, although the decomposition in UQC does result in

Figure 3.13: Execution time of the decomposition and circuit generation for OpenQL, Qubiter and UQC for different sizes of unitary matrices.

fewer gates, it also takes about 500 times as long as decomposition in OpenQL.

3.8. CONCLUSION

With the implementation of unitary decomposition, OpenQL can now be used for any quantum algorithm that uses arbitrary unitary gates. One such algorithm is QiBAM [167], which cannot be implemented without unitary decomposition.

The decomposition generates more gates than the theoretical minimum, but the structure of the decomposition means that further optimizations can be easily integrated with the current implementation. The decomposition is performed using Quantum Shannon Decomposition, which is up to 10 times more efficient in the number of generated gates than Qubiter and only 50% less efficient than the implementation of UQC. Two optimizations were implemented to take advantage of the internal structure of the input or intermediate unitary matrices, which can drastically reduce the length of the resulting circuit. With these optimizations, the final resulting gate count can be much lower than the illustrated worst case numbers.

The decomposition results in $O(\frac{3}{4}4^n)$ CNOT gates and $O(\frac{9}{4}4^n)$ total gates. Although the execution time of the decomposition is $O(8^n)$ for matrices of size $2^n \times 2^n$, for the decomposition of up to 10-qubit gates, our implementation is 10–100 times faster than Qubiter and about 500 times faster than the implementation in UQC.

There are several avenues that can further bring down the number of gates the decomposition generates, which are as follows:

- The implementation of a minimum two-qubit circuit, such as the one described in [173] using the method from [76], if applicable;
- Additionally, the implementation of a universal three-qubit gate, such as the one in [195];

- Implementing the multiplexed R_z gate with a CZ gate, as expressed in [173];
- Reworking the QSD so that the intermediate matrices cancel out, as the input matrix has fewer degrees of freedom than the matrices resulting from the QSD. Therefore, it might be possible to choose some of these intermediate matrices in such a way that they can be decomposed using fewer elementary gates;
- The implementation of other specific efficient decompositions, such as controlled unitary gates (as opposed to uniformly controlled gates), quantum multiplexers or specialized multi-controlled rotation gates.

The implementation can also be updated to use the block-ZXZ based decomposition method presented in Chapter 2, which can be optimized to generate a circuit with fewer CNOT gates than is possible with QSD. Both decompositions have the same high-level structure, which means that the circuit-level and execution time optimizations presented in this chapter will still provide the same, or similar, benefit with the new decomposition method.

4

Efficient Parameterized Compilation for Hybrid Quantum Programming

Near-term quantum devices have the potential to outperform classical computing through the use of hybrid classical-quantum algorithms such as variational quantum eigensolvers. These iterative algorithms use a classical optimizer to update a parameterized quantum circuit. Each iteration, the circuit is executed on a physical quantum processor or quantum computing simulator, and the average measurement result is passed back to the classical optimizer. When many iterations are required, the whole quantum program is also recompiled many times.

We have implemented explicit parameters that prevent recompilation of the whole program in the quantum programming framework OpenQL, called OpenQL$_{\text{PC}}$. These parameters improve the compilation and therefore total run-time for hybrid quantum-classical algorithms. We compare the time required for compilation and simulation of the MAXCUT algorithm in OpenQL$_{\text{PC}}$to the same algorithm in both PyQuil and Qiskit. We show that efficient handling of parameterized circuits results in up to 70% reduction in total compilation time for the MAXCUT benchmark, and leads to a reduced total execution time. With OpenQL$_{\text{PC}}$, compilation of hybrid algorithms is also faster than either PyQuil or Qiskit.

This chapter is based on the following article:

- Anna M. Krol, Koen Mesman, Aritra Sarkar, and Zaid Al-Ars. "Efficient Parameterised Compilation for Hybrid Quantum Programming". In: *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. vol. 2. 2023, pp. 103–111. DOI: 10.1109/QCE57702.2023.10192

CODE AVAILABILITY

OpenQL can be found at: https://github.com/QuTech-Delft/OpenQL and the specific version and branch that were used to implement efficient parameterised circuits can be found here: https://github.com/anneriet/OpenQL. The code implementations of the MAXCUT benchmark in OpenQL, OpenQL$_{\text{PC}}$, PyQuil and Qiskit can be found at https://github.com/anneriet/efficient_params_code.

4.1. INTRODUCTION

Even though Google announced quantum supremacy in 2019 [11], universal, fault-tolerant quantum computers are still a thing of the future. In the meantime, Noisy Intermediate-Scale Quantum (NISQ) devices like the Google Sycamore Quantum Processing Unit (QPU) have the potential to outperform classical computers in specific cases [25].

The NISQ era means that anybody writing quantum algorithms has to contend with a limited number of qubits and a trade-off between circuit depth and error-rates. The demonstration from Google on a 53-qubit chip had a fidelity of only 0.2%, for example [11].

Quantum algorithm development in the NISQ era is largely done with simulations of quantum devices, which are more readily available and still faster than real quantum devices. And many algorithms require more (interconnected) qubits than the current state-of-the-art has to offer. In addition, simulators offer other advantages, such as access to the full state of all qubits, error-free execution, setting of specific error rates and repeatability of "random" results [101].

One such area of quantum algorithm development is hybrid quantum-classical algorithms. These are expected to be the first algorithm candidates that will result in a practical application for quantum computation [57]. Current quantum computers have too few, too error-prone qubits to be sufficient to implement purely-quantum algorithms such as Shor's factorisation algorithm [177] or Grover's search algorithm [73]. But with hybrid algorithms, some of the processing is done on a classical computer, so quantum circuits with less qubits and lower depth are required for the quantum device and more fine-grained error correction can be applied [57].

VQE, and other variational hybrid algorithms, require many iterations of the same quantum circuit. For each iteration, a set of parameters is updated according to some classical (optimization) algorithm [57]. An example of the program flow for such algorithms is shown in Figure 4.1.

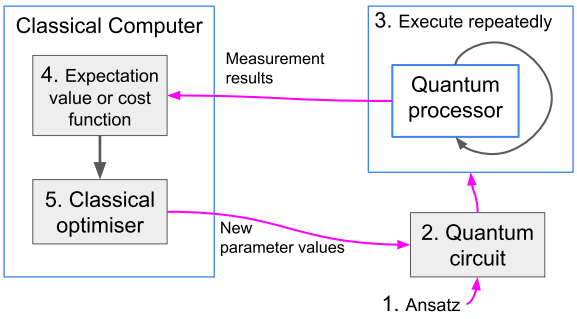


Figure 4.1: Programming flow for hybrid quantum algorithms like VQE

Since compilers for quantum programming languages require a lot of processing to produce an executable quantum circuit, doing a full compilation in each iteration consumes significant amounts of time. For iterative hybrid algorithms, however, most of

the circuit stays the same between iterations, which means that error-correction mechanisms, optimizations, decompositions, mapping, etc. are not affected. For VQE specifically, only some angles for rotation gates are changed, which means that most compilation steps are not affected for parameterized gates.

With the increasing number of qubits quantum computers can support, and with increasing qubit quality, the depth and complexity of executable quantum circuits will increase. This in turn means that the amount of time spent during this recompilation step will continue to increase, making it necessary to optimize it as much as possible. At the same time, efficient classical compilation and simulation of quantum algorithms are essential tools in the NISQ era and beyond. For as long as quantum supremacy has not yet been reached, classical computers will continue to do part of the work.

Additionally, each QPU requires a classical control system, for data conversion and implementation of QPU instructions. This classical device controls the qubits directly through analog systems. When QPUs are integrated into High Performance Computing (HPC) nodes, an additional latency bottleneck is introduced by the movement of data across the complete stack of the HPC and quantum computing system [87]. This bottleneck can be improved by reducing the amount of data that needs to be transferred to the QPU during the execution of hybrid algorithms [43].

In this chapter, we introduce OpenQL Parameterized Compilation ($\text{OpenQL}_{\text{PC}}$) which reduces compilation time and latency of hybrid quantum algorithms. The contributions of this chapter are as follows:

- A more efficient compilation process for parameterized hybrid quantum programming
- Implementation of our method in the OpenQL quantum programming framework
- Implementation of the MAXCUT quantum programming benchmark in $\text{OpenQL}_{\text{PC}}$

This chapter is structured as follows. A background on VQE and utilisation of parameters in programming languages is given in Section 4.2. Then the design goals are presented in Section 4.3. The compilation process and improvements thereof can be found in Section 4.4. After that, the methods and results are discussed in Sections 4.5 and 4.6. The conclusion can be found in Section 4.7.

4.2. BACKGROUND

To demonstrate the effect that a more efficient compilation of parameters can have, the VQE algorithm will be used. We will compare our implementation, $\text{OpenQL}_{\text{PC}}$, against Qiskit (0.37.0) and PyQuil (3.1.0), so some background on these will be given as well.

4.2.1. VARIATIONAL QUANTUM EIGENSOLVERS

VQE is a class of hybrid quantum algorithms, i.e. it uses both classical and quantum resources, to find solutions to eigenvalue and optimization problems. With VQE, quantum devices with as few as 40-50 qubits might outperform purely classical approaches [149]. VQE can run on any gate-model quantum device, is able to leverage the strengths of a

specific architecture and to variationally suppress some errors [131]. To do this, it uses a parameterized quantum circuit, where the parameters are updated variationally according to a classical optimization algorithm.

The execution flow of VQE is shown in Figure 4.1. VQE can be used to determine the ground state and the ground state energy of a Hamiltonian H.

To do this, a parameterized ansatz is used. The parameter values re adjusted with each iteration of the circuit until convergence. The final parameter values determine the ground state of the Hamiltonian [131].

There are many options for the choice of ansatz, as well as the choice of classical optimizer. The ansatz can be tailored to many aspects of the algorithm and system used, such as the specific hardware implementation [131], specific problems, accuracy or circuit depth [72]. Different classical optimizers converge at different rates and respond differently depending on the amount of noise present in the quantum system [148].

The number of iterations before algorithm convergence depends on many different details. To give an indication, the qubit efficient implementation of VQE in [124] reaches the ground state in 500 iterations, with 17-qubit circuits that contain 180 to 900 variational parameters. The various VQEs from [98] are 6-qubit circuits with 30 parameters each done for 250 iterations, with 10^3 measurements per iteration to estimate the expectation value.

4.2.2. PROGRAMMING OF PARAMETERIZED CIRCUITS

In this chapter, we compare OpenQL$_{PC}$ against Qiskit and PyQuil, quantum programming languages from IBM and Rigetti, respectively. Both allow programming of parameterized circuits, are widely used and can be used as a library from a Python program. This makes them a good comparison to OpenQL$_{PC}$, which also has these features.

The aim of OpenQL$_{PC}$ is to be easy to use for new quantum programmers as well as for people already familiar with these other languages. We will therefore give an overview of how parameters can be used in these other languages and aim to make our syntax similar so that adopting the new feature in OpenQL will be simple.

QISKIT

Qiskit is the quantum framework of IBM. It is suited for working with noisy qubits, which can be simulated using their simulator, Qiskit Aer. This allows classical simulation of circuits compiled using the Qiskit compiler, Qiskit Terra. Besides simulation, circuits compiled using Qiskit Terra can also be executed on real quantum devices using IBM Q [153].

The following example illustrates the use of parameterized circuits in Qiskit:

```
1 qcirc = QuantumCircuit(2)
2 theta = Parameter("theta")
3 pvec = ParameterVector("pvec", 2)
4 qcirc.ry(theta, 0)
5 qcirc.crx(pvec[1], 0, 1)
6 qcirc.assign_parameters({pvec[1]: 1]}, inplace=True)
7 theta_range = np.linspace(0, 2*np.pi, 128)
8 circuits = [qcirc.bind_parameters({theta: theta_val}) for theta_val in
      theta_range]
9 qcirc.qasm(filename="output.qasm")
```

Defining parameters in Qiskit can be done individually, as shown above on line 2, or as a vector with a specified length, as on line 3. Both can be used as gate arguments, as on lines 4 and 5 of the example. Binding parameters to values can be done using the command `assign_parameters` (line 6) to generate a single circuit, or by `bind_parameters` to generate a list of circuits with a circuit for each value of the parameter. This is shown on line 8. This code generates a list of 128 circuits, one for each of the values of `theta` as defined on line 7. The circuit can be output as OpenQASM to a file, as shown on line 8.

The circuits can be run on a real quantum system by compiling them individually for a specific backend. An example is shown here of how to run the circuits in the example above:

```
1 compiled_circuit = transpile(circuits[64], simulator)
2 compiled_circuit = assemble(circuits, simulator)
3 job = simulator.run(compiled_circuit, shots=10)
4 counts = job.result().get_counts()
```

On line 1, one of the circuits is transpiled to be executable on the `simulator` backend. A list of circuits can also be compiled all at once, as on line 2. In both cases, the resulting `compiled_circuit` can be executed on any supported backend with the execute command, as shown in line 3. Any measurement results can be retrieved by calling `result().get_counts()` as on line 4. This gives the measurement results as counts of how often each possible bit combination was measured, for example: `{'0': 4, '1': 6}`.

Compilation, execution and binding of parameter values can also be combined into a single `execute` command, as shown below:

```
1 job = execute(compiled_circuit,
2               backend=QasmSimulator(),
3               parameter_binds=[{theta: theta_val} for theta_val in
    theta_range])
```

Using a single command to bind parameters and simulate the circuit makes it impossible to determine the individual execution time of the components from the host Python program. So to get those results the separate commands are used. The compilation is considered complete after the `assemble` command.

PyQuil

PyQuil is the quantum programming language from Rigetti. It is compiled using the Quil compiler into Quil, also the name of their Quantum Instruction Language [180]. PyQuil can be used to write hybrid algorithms with parameters [99], which makes it a suitable comparison to OpenQL$_{\text{PC}}$.

A PyQuil program with the same overall functionality as the Qiskit example is shown below. The program will be used to explain how parameters can be defined and used in PyQuil [179]:

```
1 program = Program()
2 ro = program.declare('ro', memory_type='BIT', memory_size=1)
3 theta = program.declare('theta', memory_type='REAL')
4 program += RX(theta, 0)
5 program += MEASURE(0, ro[0])
```

A quantum program is defined on line 1. Two types of parameter are declared on lines 2 and 3; `ro` will be used to store the measurement results, and the parameter `theta` is used as argument for the `RY` gate on line 4.

The parameterized quantum program `program` from this example can be run for different values of `theta`, as shown in the following code listing [39]:

```
1 parametric_measurements = []
2 executable = simulator.compile(program)
3 for theta_val in np.linspace(0, 2 * np.pi, 128):
4     executable.write_memory(region_name='theta', value=theta_val)
5     result = simulator.run(executable)
6     parametric_measurements.append(result)
```

In line 1, the array `parametric_measurements` is defined for storing the measurement results. The `program` from the previous example is compiled for a specific execution platform, `simulator` in this case. A for-loop is used to iterate over the values of `theta_val` (lines 3-6). On line 4, the value of `theta_val` is written to memory region `theta`. The program is then run on a simulator and the measurement result is appended to the list `parametric_measurements` on lines 5 and 6.

4.3. DESIGN GOALS

Implementation of parameters in OpenQL$_{PC}$ was done with the following design goals in mind: modularity, scalability, user-friendliness (usability), future-proof and speed.

Modularity: To make OpenQL$_{PC}$ robust against future changes to the host programming language, the parameters will be implemented as a separate entity.

Scalability: This has two parts, the first is that OpenQL$_{PC}$ will make it easy for the programmer to define many parameters, and to allow putting values to them in bulk as well. The second part is that the overall compilation time should not be affected (too much) by the number of parameters.

Usability: The syntax and use of the parameters will be made similar to other quantum programming languages (Qiskit and PyQuil), for users already familiar with those and to make manual rewriting of a program from one language to the other easier. Clear errors should be provided, and parameter types should be explicit. Finally, the programmer should not be required to manually provide a string for each parameter, which can become cumbersome for bigger programs. But manual naming should be supported, in case human-readable common Quantum Assembly Language (cQASM) is desired.

Future-proof: Parameterisation also means a more clearly defined boundary between the static and dynamic parts of a quantum circuit, which can be used in the future for a full split between those two compiler functionalities, or for detection of parallelism. In addition, the compile speed, number of supported parameters etc. should allow this feature to be used (and useful) into a future of quantum programming languages, where extensive knowledge of quantum is no longer required to write quantum programs.

Compilation speed: OpenQL is a high-level quantum programming language with an extensive compilation toolchain [103]. It supports gate decomposition, circuit optimization, scheduling and mapping, all of which are necessary to be able to execute the generated cQASM on NISQ devices [37]. This also makes the compilation in OpenQL and all other such compilers computationally expensive, in terms of classical resources. For

algorithms that require only a single compilation pass, the classical compilation time is negligible compared to execution of a quantum program on a real quantum device, or classical simulation of the circuit. However for iterative hybrid algorithms, such as VQE, repeated compilation of essentially the same quantum circuit becomes a much bigger drain on resources. With explicit definition of the dynamic parts of the circuit through the use of parameters, the bulk of the compiler operations does not need to be repeated for each iteration of the hybrid algorithm.

4.4. PARAMETERISATION IN OPENQL$_{PC}$

The first time a quantum program is compiled (by calling compiler.compile(..)), the full stack is executed, including gate decompositions, mapping, optimizations, etc. [103]. Without using our OpenQL$_{PC}$ approach, updating of any (parameter) values requires repeated execution of this whole stack, as shown with the grayed-out arrows in Figure 4.2.

4.4.1. COMPILE FLOW

With OpenQL$_{PC}$, only the affected gates are updated with the numerical values of the parameters as shown in Figure 4.2. This means that there is no unnecessary repetition of the whole (extensive) compiler stack for every iteration of an iterative quantum algorithm.

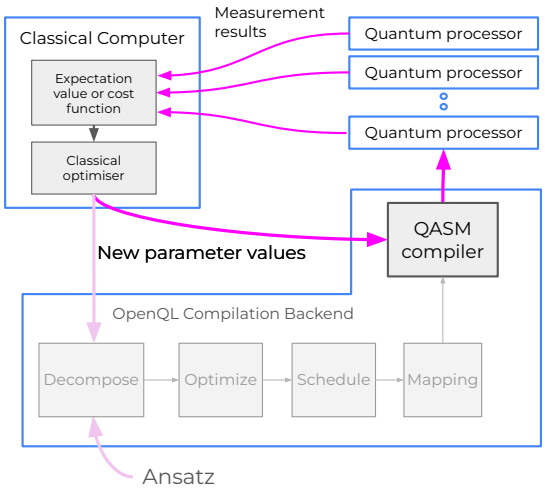


Figure 4.2: Compilation flow of OpenQL with and without using the new parameter implementation [103, 169]

In the first compile run, any parameterized quantum instructions without a numerical value are skipped by any compiler passes that require that specific value. Mapping and scheduling one (or a set of) instruction cannot be done when the qubit is not specified, but do not require knowledge of the angle of a rotation gate. When recompiling, only parameterized quantum instructions and affected instructions are considered, while the rest of the code is left unaffected. This might mean some optimizations that

Table 4.1: Parameter syntax

[openql.openql.] Param(*Type* [, *Name*] [, *Value*])
Type : "INT" | "REAL" | "ANGLE"
Name : Symbolic name of the parameter, will appear unmodified in resulting cQASM (string)
Value : Numerical value, must match the type as specified by *Type*

affect large regions of the code will not be done, but this effect is expected to be small compared to the resources saved by not checking all of the code after each compilation.

4.4.2. IMPLEMENTATION

In this section, we will explain and document how parameters can be used in OpenQL$_{PC}$in more detail.

Before a parameter can be used it needs to be created, and at some point it will need a (numerical) value. Assigning a numerical value to the parameter can be done at construction, individually at any point in the program or at compile time. All this is explained in more detail below.

PARAMETER CONSTRUCTION

Parameter construction requires the specification of a type, which can be one of "INT", "REAL" or "ANGLE". "REAL" and "ANGLE" are essentially the same, both map to an underlying "double" type. Depending on the type, a parameter can substitute hard coded qubit numbers or gate angles. Optionally, the user can specify a name for the parameter at construction or directly assign a value to it. The syntax for parameter construction can be found in Table 4.1. In the code listing below, example code is shown for parameter construction with specification of 1. parameter type only, 2. type and parameter name, 3. type and numerical value, and 4. type, parameter name and numerical value.

```
p_int = ql.Param("INT")
p_real = ql.Param("REAL", "pname")
p_angle = ql.Param("ANGLE", 1.724)
p_int2 = ql.Param("INT", "pname2", 4)
```

USING PARAMETERS

Parameters can be used in quantum circuits, in place of hard coded qubit numbers or gate angles. Some examples of using parameters are shown below, where the parameters are as defined in Section 4.4.2:

```
kernel.gate("hadamard", p_int)
kernel.gate("rz", [0], 0, p_real)
kernel.gate("ry", p_int2, p_angle)
kernel.gate("cnot", p_int, p_int2)
```

On line 1 a "hadamard" gate is added to the `kernel`, where the parameter `p_int` is used in place of the qubit number. On line 2, parameter `p_real` is used as the rotation angle of the "rz" gate. Both of these can be combined, as on line 3, where the "ry" gate is applied to the qubit number stored in parameter `p_int2` and with the angle stored in `p_angle`. Parameters can also be used for 2-qubit gates, as shown on line 4, where a `cnot` is applied to qubits `p_int` and `p_int2`.

COMPILING PARAMETERS

Quantum programs with parameters can be compiled in the same way as circuits without parameters in OpenQL, as shown in the code listing below:

```
1 compiler.compile(Program)
```


When the `kernel` from Section 4.4.2 is added to a program and compiled in this manner, the resulting cQASM output is shown below:

```
1     hadamard %w5Bq1DRO
2     rz q[0] %pname
3     ry q[4], 1.724
4     cnot %w5Bq1DRO, q[4]
```

Line 1 shows the `hadamard` gate from Section 4.4.2 with parameter `p_int`. Symbolic names in the cQASM code are preceded by the % symbol, and the randomly generated string "*w5Bq1DRO*" is the "name" of this parameter. Line 2 shows the `rz` gate, applied to qubit 0. The angle is the parameter `p_real` from Section 4.4.2. The name of this parameter was set as `pname` at construction, and this is reflected in the cQASM output. On line 3, the `ry` gate had `p_int2` in place of a qubit number and `p_angle` in place of a rotation angle. Both were constructed with a numerical value already set, which is reflected in the cQASM output above, where instead of symbolic names the numerical values are used; qubit number 4 (`q[4]`) for `p_int2` and 1.724 for `p_angle`

SETTING PARAMETER VALUES

There are three ways to set the value of a parameter in OpenQL$_{PC}$. These are:

- at construction: `p1 = Param(string, value)`,
- at any point individually:
  `Param.set_value(value)`, and
- at compile time: `Compiler.compile(program, [p1], [value])`

Setting a value at construction is outlined in Section 4.4.2, examples for the other two ways are given here.

Assigning a numerical value to a single parameter can be done at any point in the code by calling the `set_value` method. This is shown in the code below, where the `p_int2` is defined as in Section 4.4.2.

```
1 p_int2.set_value(2)
```

This assigns the value of 2 to `p_int`. It is also possible to modify the numerical value of a parameter in this way.

When compiling and generating cQASM, the final value of a parameter will be used for all instances of a parameter, so it is not possible to assign different numerical values to a single parameter partway through a quantum circuit. The intended use is to modify parameter values between different iterations of a circuit, where the whole circuit is compiled for each iteration.

It is also possible to set values at compile time. This can be done for all parameters at once, or for a subset of the parameters. An example using the program from Section 4.4.2 is shown below:

```
compiler.compile(program, [p_int, p_real, p_angle], [1, 2.1, -1.7])
```

With this line of code, the values of 1, 2.1 and -1.7 are assigned to parameters `p_int`, `p_real` and `p_angle`, respectively, and the whole program is compiled.

This results in the following cQASM output:

```
1     hadamard q[1]
2     rz q[0], 2.1
3     ry q[4], -1.7
4     cnot q[1], q[4]
```

On line 1, the `hadamard` gate from before, with parameter `p_int`, is applied to qubit 1 (q[1]), the value stored in `p_int`. This overwrites the value set in Section 4.4.2. The `rz` gate on line 2 now uses an angle of 2.1, the value from `p_real`. The `ry` gate on line 3 is applied to qubit 4 as in Section 4.4.2, since the value of `p_int2` was not modified. The angle is now `-1.7`, the value assigned to `p_angle` at compilation. On line 4, the `cnot` gate is applied from qubit 1, as stored in `p_int` and to qubit 4, as stored in `p_int2`.

The resulting cQASM no longer contains symbolic names, and can now be executed on a simulator or a real quantum device.

4.5. COMPARISON TO OTHER PROGRAMMING LANGUAGES

To determine the influence of parameters and to be able to compare the implementation to other programming languages, execution time tests were performed. These were done with the MAXCUT benchmark [133], which was implemented in OpenQL, OpenQL$_{PC}$, Qiskit and PyQuil.

4.5.1. THE MAXCUT BENCHMARK

MAXCUT is a hybrid algorithm which can be used for circuit layout design [17], statistical physics and more [200]. It aims to find the division of a graph into two parts, where the total weight of all edges between the parts is maximized [200]. The general case is an NP-hard problem.

The MAXCUT benchmark from [133] was implemented in OpenQL, PyQuil and Qiskit on 4-regular graphs of varying number of nodes. These graphs and this benchmark were chosen because a reference implementation was already available, it can be easily scaled up to any number of nodes and the number of qubits and the circuit depth both increase

linearly with an increasing number of nodes. It is a variational quantum algorithm, with an execution flow as in Section 4.2.1.

4.5.2. EXPERIMENTAL SETUP

In order to compare the performance of OpenQL$_{PC}$ with other programming languages, the MAXCUT algorithm was implemented in OpenQL, OpenQL$_{PC}$, Qiskit (0.37.0) and PyQuil (3.1.0). Timing was done from the host Python program using the "time" package, on a Dell Latitude 7400 with an 8th Generation Intel Core™ i7-8665U Processor and 2x 4GiB DDR4 RAM.

First, the compilation time was measured with minimal influence of outside factors. To this end, the MAXCUT benchmark was run for each programming language with a constant problem size of 3 steps and 15 nodes, while the number of iterations was varied from 1 to 16. To avoid any influence of the classical optimizer on the measurements, the parameter values for each iteration were randomly generated outside the timing loop.

Then, to determine the performance of OpenQL$_{PC}$ for the real benchmark, the compilation and the total execution time were measured for the full benchmark for varying problem sizes. The number of function evaluations for the classical optimizer was limited to 100, and any runs that reached convergence earlier were discarded.

To limit the influence of other factors on the measurement results, the angles for each iteration were not generated using a classical optimizer, but randomly generated outside of the timing loop. The number of steps was set at 3, as mentioned before, and the number of nodes at 15. This corresponds to a circuit with a combined total of 330 CNOTs and rotation gates, and 15 measurement operators. For each separate measurement, the language (OpenQL, OpenQL$_{PC}$, Qiskit and PyQuil) and the total number of iterations are randomly selected. For the iterations, a simple loop is used that compiles the circuit with different angle values for the parameters. To make sure we include all of the compilation steps but as few other steps as possible, the wall-clock starting time is measured just before the first line where the quantum programming language was used. Some programming languages use an ASAP strategy, while others only start the actual compilation when a "compile" instruction is called. To verify that actual circuit generation took place for each iteration, (Open)QASM files are generated as output. All random number generation and other preparation steps are done before the starting time, and all handling and writing of measurement data after the ending time. The wall-clock ending time is measured after all the iterations have completed, and total time is taken as the difference between the ending and the starting time.

Timing was done for a total of 1, 2, 4, 6, 8, 10, 12, 14 and 16 iterations. Although the circuit that is used in these tests comes from the MAXCUT benchmark, no classical optimizer is used for this set of tests. This way, the number of iterations is not dependent on any unknown factors, and the compilation time does not include the runtime of the optimizer. The compilation of the circuits includes an optimization step to get more realistic results and to better show the advantage that parametric compilation can offer. With parametric compilation, the circuit will only be optimized (and decomposed, scheduled and mapped) once, instead of every iteration. The Qiskit optimization level was set to 1, which corresponds to "light optimization". This adds adjacent gate collapse and redundant reset removal to the compilation stack [153]. The closest equivalent in

OpenQL is the "RotationOptimizer" pass, which tries to find sets of contiguous gates that correspond to identity, and then takes those out of the circuit. Both of these optimizations handle only small sets of adjacent gates, and remove or collapse them when possible [102].

To determine whether the improvements made have an impact on any real applications, a full implementation of MAXCUT was used. The circuit and the parameters are the same as in the previous set of tests, but now a classical optimizer was added to the loop to determine the parameter values for the next iteration. The circuits were also run on a simulator, and the measurement results coming from the simulator are used as input for the optimizer. This makes the total execution flow as in Figure 4.1. The tests were performed for regular graphs with between 3 and 8 nodes, with 3 steps to the algorithm as before. Tests were performed interleaved where possible, and number of nodes was randomized for each trial. For each run of the MAXCUT benchmark, the number of function evaluations was limited to 100, and any runs that reached convergence earlier where discarded. Therefore the total number of function evaluations for each language is 100, although the number of circuit compilations and simulations can be lower. This is because the loop is aborted preemptively if the optimizer generates negative angles or angles bigger than 2π. The effect this has on the results is expected to be small, and it should be the same for each language so does not influence the comparisons made.

4.6. EXPERIMENTAL RESULTS

The results for the first set of tests outlined in Section 4.5.2 can be found in Figure 4.3. The figure shows the wall-clock time in seconds that it cost to compile for the different numbers of iterations for OpenQL$_{PC}$, OpenQL, Qiskit and PyQuil. Figure 4.4 shows these measurements relative to the time of a single iteration (i.e., divided by the time it costs to do just one iteration).

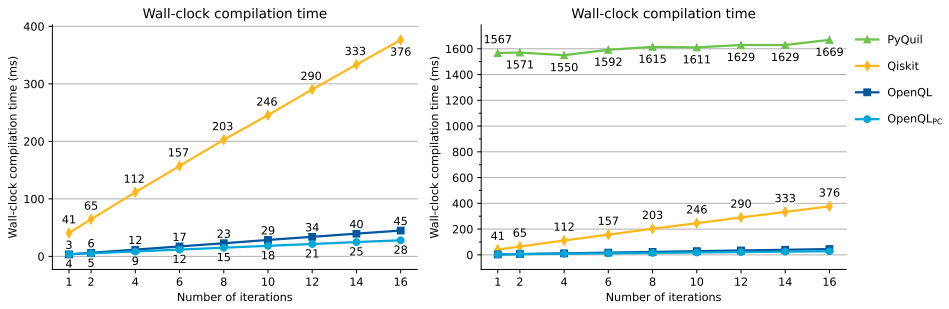


Figure 4.3: Wall-clock compilation time for a MAXCUT circuit with 15 nodes, run without optimizer, with varying number of iterations for OpenQL$_{PC}$, OpenQL, Qiskit and PyQuil

As can be seen in Figure 4.3, both OpenQL$_{PC}$ and OpenQL are considerably faster than Qiskit and it is clear that all are much faster compilers than PyQuil. When considering the relative increase in compilation time, Figure 4.4 shows that OpenQL handles repeat compilations worse than either Qiskit, OpenQL$_{PC}$ or PyQuil, since it is not

optimized for this operation. In OpenQL, it takes 13 times as long to do 16 iterations compared to a single iteration. This is because of setting up the circuit and the compiler, which are done for each iteration instead of only once for the complete run in OpenQL$_{PC}$. The time that is saved by OpenQL$_{PC}$ for subsequent iterations can be seen in both absolute and relative decrease in compile time. OpenQL and OpenQL$_{PC}$ take almost the same amount of time for a single compilation, but for each subsequent iteration, there is more and more time saved by OpenQL$_{PC}$.

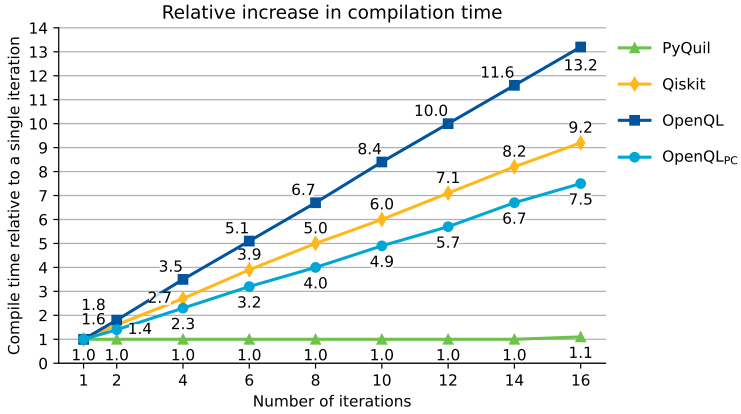


Figure 4.4: Compilation time relative to the compilation time of a single iteration

Looking at the absolute compile times shows that Qiskit is an overall much slower compiler than OpenQL or OpenQL$_{PC}$. Inspecting the relative cost of compiling for multiple iterations, Qiskit shows some level of optimization for this type of repeated compilations. Still these optimizations are not as effective as the ones implemented in OpenQL$_{PC}$, which is faster both in absolute terms and in relative cost of compilation.

Inspecting the compilation time of PyQuil, Figure 4.4 shows that consecutive iterations do not incur much additional time. However, as can be seen in Figure 4.3, the compile times of PyQuil are 10x to 100x higher than that of the other measured languages. This might be because PyQuil has to be compiled by running a separate compiler on a virtual machine in server mode, which is called from the main python program.

In Figure 4.5, we also inspect the accumulated compilation times for running MAX-CUT with a classical optimizer for graphs of different numbers of nodes, for OpenQL$_{PC}$, OpenQL, Qiskit and PyQuil. OpenQL$_{PC}$ has the shortest total compile time, second is Qiskit, then OpenQL, and the slowest by an order of magnitude is PyQuil. These benchmark runs were done with fewer nodes and more iterations (namely 100 iterations) than the earlier tests, and it is interesting to see that OpenQL has overtaken Qiskit in compile time. Although Qiskit takes considerably longer than OpenQL to compile a circuit once, as noted before, the time that is saved for each subsequent iteration results in a total shorter time spent compiling. The compile time of PyQuil shows that it is actually the circuit length or the number of qubits that lead to such high compile times, as the circuit with 3 nodes takes 614 ms to compile for 100 iterations, and the 15 node circuit

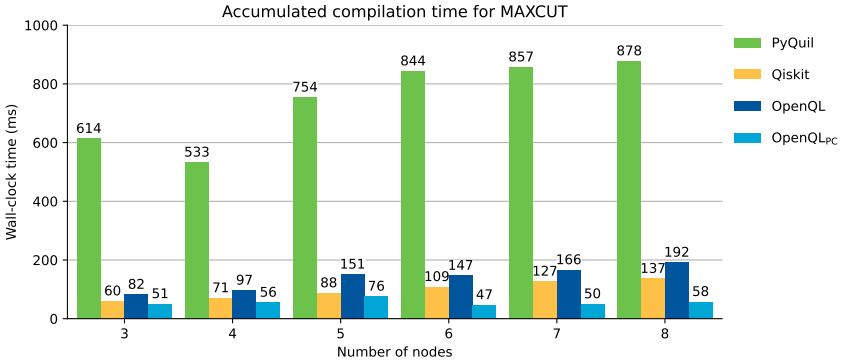


Figure 4.5: Accumulated compilation time for the MAXCUT benchmark with 100 iterations for graphs with 3 to 8 nodes, for OpenQL$_{PC}$, OpenQL, Qiskit and PyQuil

from Figure 4.3 takes 1567 ms for a single iteration. So although PyQuil uses little time for compiling subsequent iterations, it is most likely just the first compilation pass of a circuit that takes such a long time. OpenQL$_{PC}$ combines the already fast compile times of OpenQL with efficiently handling iterations, and as a result is much faster than all other tested options.

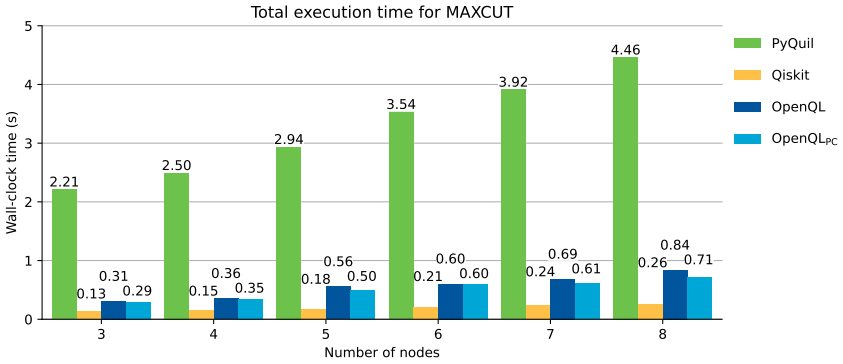


Figure 4.6: Total execution time for the MAXCUT benchmark with 100 iterations for graphs with 3 to 8 nodes, for OpenQL$_{PC}$, OpenQL, Qiskit and PyQuil

Complete (simulated) runs of MAXCUT were also timed, to determine the influence of the compiler optimization on the total runtime, and to verify that our earlier results do not include any simulation. These are plotted in Figure 4.6. If any of the previous tests included simulation, we would expect the measured compile times to be close to these execution times. Since this is not the case, none of the tested compile times included simulation.

The figure shows that total execution times are up to 50x higher than just compilation times (Figure 4.5). So although the main difference in execution time between the pro-

gramming languages is due to the difference in performance between the corresponding simulators. Still, the same trend can be seen, where PyQuil takes the longest time by far, and both OpenQL and OpenQL$_{PC}$ have similar performance, since OpenQL$_{PC}$ and OpenQL both use the QX simulator. The figure also shows that the Qiskit Aer simulator is the fastest of the tested options.

In summary, the results show that the improvements made for OpenQL$_{PC}$ result in a clear speedup compared to OpenQL. Compared to Qiskit and PyQuil, OpenQL$_{PC}$ has the fastest compile times for the MAXCUT benchmark. However, when looking at the total execution time, the faster simulation by Qiskit Aer results in the shortest total execution time of the benchmark. This can be partly explained by the large communication time between OpenQL$_{PC}$/OpenQL and the QX simulator, which requires writing and reading of a QASM file for every iteration. This is especially apparent in the MAXCUT benchmark, which has a lot of iterations for relatively short circuits, which results in a lot of read/write operations compared to circuit compilation(s). Between the Qiskit compiler and the Qiskit Aer simulator, however, the circuit can be passed directly.

4.7. CONCLUSION

In this chapter, we introduced OpenQL$_{PC}$, an efficient approach to parametric compilation for hybrid quantum-classical algorithms, and implemented it in the OpenQL programming framework. OpenQL$_{PC}$ is designed to be modular, scalable, usable, future-proof and fast. We compared wall-clock compilation time of OpenQL$_{PC}$ with OpenQL, Qiskit and PyQuil. The total compilation time was measured using the MAXCUT benchmark from [133].

Experimental results show that compared to other programming languages, total compile time of OpenQL and OpenQL$_{PC}$ are between 10 and 20 times faster than Qiskit, respectively. PyQuil has the slowest compilation, about 60x longer than OpenQL$_{PC}$. In addition, comparing compile times of multiple compile iterations relative to single iterations shows that OpenQL$_{PC}$ is the fastest, followed by Qiskit which is 1.2x slower, while OpenQL took the most time being 1.8x slower than OpenQL$_{PC}$.

The MAXCUT benchmark was also implemented in its entirety, including a classical optimizer and simulations. These tests, with shorter circuits but more iterations than before, show that the improvements made in OpenQL$_{PC}$ result in a decrease of 40 to 70% in (accumulated) compilation time. This makes OpenQL$_{PC}$ faster than all other tested languages. For the total execution (run)time of the MAXCUT benchmark, the performance of the simulators has more influence than the compile times of the languages. The simulator used with PyQuil is still the slowest option by far, but the Qiskit Aer simulator is 2 to 3 times as fast as the QX simulator used by OpenQL and OpenQL$_{PC}$. Even so, the faster compile time of OpenQL$_{PC}$ does lead to a clear speed-up of the complete benchmark.

Furthermore, a single compilation of a quantum circuit is projected to become more computationally expensive as more sophisticated mapping, optimization and error-correcting algorithms are created and implemented, which will further increase the cost of repeated compilations in hybrid algorithms and increase the impact of our approach.

Our approach can be extended to decrease the data transfer for hybrid algorithms, which will improve the latency bottleneck caused by the movement of data over the entire stack of HPC and QPU systems. Because we define the static and dynamic parts of

an algorithm explicitly, only the updated parameter values will have to be transferred to the classical control systems of the QPU. But to implement such an extension requires explicit access to these classical control systems. This was not available at the time of writing, although there were existing proposals to provide the required access [159, 127].

After this work was completed, Amazon announced Braket hybrid jobs, which also use parametric circuits to speed up the execution of hybrid quantum circuits [36]. This is also illustrated in Figure 4.7. This paper and Amazon Braket hybrid jobs were both presented at the QCE23 IEEE conference in Seattle.

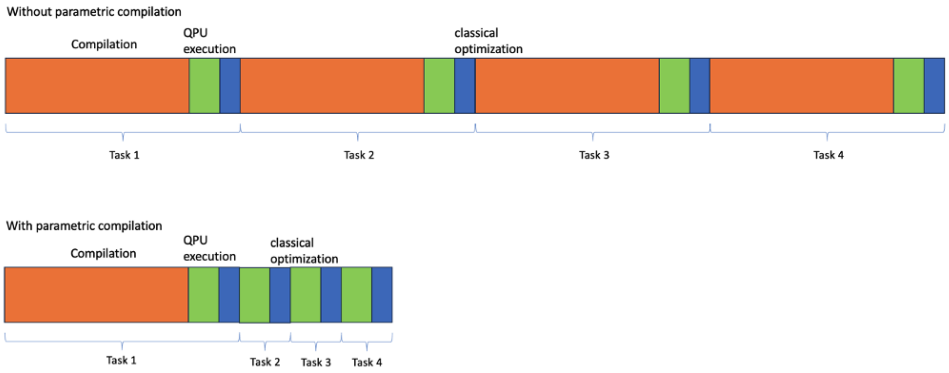


Figure 4.7: AWS breakdown of runtime for a hybrid workload on Amazon Braket, with and without parametric compilation [36].

5

QISS: Quantum Industrial Shift Scheduling Algorithm

In this chapter, we show the design and implementation of a quantum algorithm for industrial shift scheduling (QISS), which uses Grover's adaptive search to tackle a common and important class of valuable, real-world combinatorial optimization problems.

We show how QISS can be used to find the optimal schedule for n days out of a solution space of size $N = 4^{2n}$. The optimal solution is reached within 99% of cases after $\sqrt{N} = 4^n$ applications of Grover's oracle, which requires a total of $11n + 11 + \lceil \log_2(19n) \rceil$ qubits for scheduling n days. We give an explicit circuit construction of the oracle, incorporating the multiple constraints present in the problem, and detail the corresponding logical-level resource requirements. Further, we simulate the application of QISS to specific small-scale problem instances to corroborate the performance of the algorithm.

Our work shows how complex real-world industrial optimization problems can be formulated in the context of Grover's algorithm, and paves the way towards important tasks such as physical-level resource estimation for this category of use cases.

This chapter is based on the following article:

- Anna M. Krol, Marvin Erdmann, Rajesh Mishra, Phattharaporn Singkanipa, Ewan Munro, Marcin Ziolkowski, Andre Luckow, and Zaid Al-Ars. *QISS: Quantum Industrial Shift Scheduling Algorithm.* Submitted to *IEEE Transactions on Quantum Engineering*. 2024. arXiv: 2401.07763 [quant-ph]

Code availability

The repository containing the full source-code of QISS can be found at https://github.com/anneriet/QISS. This repository also contains the code that was used to generate all the figures in this chapter. The full implementation of QISS is located in the "QISS/QISS_with_cost.ipynb" Python notebook and additionally in the "QISS/operator_definitions.py" python file.

5.1. INTRODUCTION

Many industries face complex optimization challenges too large and complex to solve optimally [20]. For example, in the automotive industry, these problems may include optimizing supply chain logistics [63, 189, 13], quality control [59, 79], robots path planning [171], and shift scheduling [192, 204]. These challenges feature numerous constraints and expansive solution domains that expand exponentially with each added variable to the problem's framework.

In particular, the shift scheduling problem in automotive production networks concerns the creation of a schedule for the numerous production steps. The goal is to maximize productivity while considering dependencies between production steps, the discrete number of possible shift durations, and working regulations and preferences. As production steps, workers, and shift configurations increase, the solution space grows exponentially.

Solving such industry-scale scenarios optimally for an extended period (e.g., one year) is not feasible, so such problems are often solved heuristically [105]. Heuristic methods can provide valid shift schedules in reasonable computation times, but cannot guarantee an optimal solution, resulting in additional costs due to over-staffing or a reduced production volume.

Quantum computing approaches for approximate solutions to such optimization problems have also been proposed [133], such as [206, 75, 183, 169] based on quadratic unconstrained binary optimization (QUBO) and quantum annealing, as well as quantum acceleration of branch-and-bound algorithms [136] and reinforcement learning [168]. Like classical heuristics, most of these algorithms have no mathematical bounds for the solution quality: they can provide feasible solutions under the right conditions (e.g., a structured solution space), but it is impossible to say how close to the optimal solution the algorithm got. It is therefore valuable to compare the approximate answer to that of an exact solver for problem sizes that can still be solved exactly.

This chapter provides the first implementation of an exact quantum algorithm for the industrial shift scheduling problem, which we call QISS. The term "industrial shift scheduling" is chosen for simplicity for a problem that is about the optimization of the schedule of factory shop operating hours, as described in Section 5.3. In other formulations of the problem one may be faced with a related version, which differs e.g. in its objective function or its constraints.

QISS is built upon the Grover's adaptive search (GAS) procedure, wherein Grover's quantum algorithm is executed iteratively with input conditions at step i derived from the output of step $i-1$. QISS then inherits the theoretical asymptotic quadratic speedup over classical algorithms for unstructured search provided by Grover's algorithm itself.

While unstructured search is not employed in practice as a method to solve industrial-scale problems, it is often used on relatively small problem instances with the goal of benchmarking the performance of heuristic algorithms. In this context, a quadratic speedup could make a significant difference to the scope for analyzing the performance of heuristics. Whether such a form of advantage can be achieved with an exact quantum algorithm such as Grover's may depend strongly on the use case at hand. To begin to answer the question, one must construct an explicit quantum circuit for the Grover's 'oracle', which is responsible for identifying solutions that satisfy some desired

search criteria.

Our work details the construction of a Grover's oracle circuit for the industrial shift scheduling problem. In particular, we show how the problem's characteristic and complex constraints, related to production targets and intermediate storage limitations, can be incorporated directly into the oracle. We expect our work to be valuable to researchers investigating quantum computing approaches to similarly highly constrained optimization problems, and in paving the way for a thorough physical-level resource estimation for this specific problem type.

The contributions of this chapter are as follows:

- We formulate a shift scheduling problem with real-world constraints comparable to industry-relevant use cases,
- We implement QISS, the first quantum algorithm that gives an exact solution to the shift scheduling problem, and
- We verify and evaluate QISS to show the correctness of the algorithm and that it can be used to solve the shift scheduling problem with quadratic speedup.

This chapter starts with a background discussion in Section 5.2 on the industrial shift scheduling problem and current approaches to the use case, including quantum computing algorithms like Grover's search. The construction, implementation and validation of the quantum industrial shift scheduling algorithm are described in Section 5.4, The chapter concludes with Section 5.6 with a summary of the findings. The accompanying code is publicly available on GitHub.

5.2. BACKGROUND

This section will give background on the industrial shift scheduling problem, and an overview of classical and quantum solutions for the use case.

5.2.1. INDUSTRIAL SETTING AND UTILITY OF THE PROBLEM

The literature on shift scheduling use cases is vast due to its relevance in almost all economic and social sectors. Examples can be found for the optimization of nurse shift schedules in hospitals [88], staffing of employees in call centers [130], retail [7, 97], or the postal service [18]. Overviews of more use cases and solution approaches show the whole spectrum of this optimization problem [146, 192].

Shift scheduling is also essential to production planning in manufacturing facilities, especially in the automotive industry. To automate the scheduling process and minimize costs while maximizing productivity, manufacturers use optimization algorithms for shift scheduling. An optimization algorithm can analyze vast amounts of data and variables, such as the production schedule, worker availability, working regulations, and dependencies, to generate a shift schedule that meets production targets, reduces labor costs, and improves worker satisfaction.

The main difference between shift scheduling in the manufacturing sector and other sectors is the added constraint of a target production volume. Unlike other versions of the shift scheduling problem, the objective is not to cover all shifts adequately. Instead,

the goal is to maximize the overall productivity of end-to-end manufacturing systems comprising multiple production sites, referred to as *shops*, while minimizing labor costs and complying with legal regulations.

One of the critical business motivations for implementing an automated shift scheduling algorithm is the significant problem size associated with industrial manufacturing facilities. Companies operate in global production networks, and each facility has its own unique production lines, teams, and shifts, making the scheduling process complex and challenging. The optimal scheduling of thousands of employees working in various roles, including production, maintenance, logistics, and administration, is highly complex and implies a high potential for reduced labor costs and more consistent production volumes.

More reliable production volumes are a crucial performance metric for manufacturing companies, especially in the automotive industry, and any disruptions to the production process can negatively affect this metric. An optimal schedule ensures that shifts are planned prudently so that an adequate number of workers is always available. Interdependent steps in the production process must be synchronized, and the limited intermediate storage between these steps, so-called *buffers*, must be managed carefully to minimize production downtime and increase the probability of reaching the production target corridor.

Depending on the industry sector and the manufacturing entity's size, the problem's scale can vary tremendously. In the automotive industry, the annual production volumes of individual vehicle models reach values of several hundred thousand units with small margins of a few percent of the overall volume.

The shift scheduling problem is known to be NP-complete [100]. The number of different shift configurations scales exponentially with the number of production steps in the process and the number of days in the schedule. Thus, the solution space of industry-scale scenarios with around ten production steps and approximately 300 working days per year is too large to be fully explored with classical methods in a reasonable amount of time, even when considering working time restrictions and regulative constraints cutting the feasible solution space considerably.

In the remainder of this chapter, unless otherwise stated, when we refer to the scheduling problem, this means the volume-constrained industrial shift scheduling problem.

5.2.2. CLASSICAL HEURISTICS

Because of its complexity and large scale in industrial applications, the volume-constrained shift scheduling optimization problem can currently only be solved heuristically. Constructive heuristics allow dividing the problem into smaller parts, solving each of them individually, updating the constraints according to the solutions found for the prior parts, and combining them in the end to a valid solution [160].

The combined solutions of the subproblems do not necessarily form a globally optimal solution for the entire problem, even if each individual subproblem can be solved optimally. An example of such a constructive heuristic based on optimal solutions of subproblems is the following: a branch-and-cut algorithm identifies the optimal shift schedule for the first week of one year [160]. The solution for this subproblem is used as

the basis for the subsequent week, the constraints and restrictions are adjusted accordingly, and the branch-and-cut algorithm searches for the optimal solution for the next week.

This process is iteratively repeated until the algorithm is unable to identify a valid solution for a given week. In this case, the search procedure is repeated for the last week with a valid solution. The previously found optimal solution is penalized such that another valid solution is used as a new basis for the subsequent week. This constructive search procedure runs until 365 consecutive days with valid schedules are found.

Even such a heuristic search procedure can take several hours of computation time, due to the unstructured nature of the solution space of the shift scheduling problem. Unstructured solution spaces are difficult to explore for optimization algorithms, because solutions with small differences in their configuration can produce vastly different objective function values. Therefore, procedures like branch-and-cut algorithms - which are based on excluding portions of the solution space that are guaranteed to not include a solution that is better than the worst possible solution in the rest of the solution space - scale badly with the size of shift scheduling problems.

The optimization of shift schedules has long been known to be hard to solve optimally. Therefore, many other heuristic algorithms have been explored for different versions of the use case in literature, such as simulated annealing, genetic algorithms [15], tabu search [55], and ant-colony optimization [77]. The employee shift scheduling problem has also been identified as an interesting use case for evaluating quantum computing algorithms and comparing their performances to classical methods [126, 20]. However, none of these studies has considered the volume-constrained shift scheduling use case, which is significantly more complex to model.

5.2.3. GROVER'S ALGORITHM

QISS uses Grover's adaptive search [31, 66] to find optimal shift schedules, which combines classical adaptive search with Grover's search algorithm. A short introduction to Grover's search is given here, while the full algorithm is described in Section 5.4.

Grover's algorithm was initially formulated as a search algorithm for unsorted databases, but has since then been used in various applications, ranging from optimization [66] to approximate search [167]. The algorithm is designed for cases where only one record satisfies a particular property. Any classical algorithm must take $O(N)$ steps because, on average, half of the N records need to be evaluated to find the target record. A quantum computer can identify the record in only $O(\sqrt{N})$ steps with Grover's algorithm, thereby giving an asymptotic quadratic speedup over a classical random search algorithm [73].

A high-level overview of the circuit implementation of Grover's algorithm is shown in Circ. 5.1. Grover's algorithm consists of the following steps [66, 73]:

1. Initialization of the system to an equal superposition of all states in the solution space. This is done by applying a Hadamard gate to each qubit.

2. An oracle that recognizes the states that are valid solutions, and multiplies their amplitudes by -1. The oracle of QISS is implemented as $O^{\dagger}\text{CNOT}(|-\rangle)O$, where

the CNOT gate is controlled by condition qubits in $|c\rangle$, which are set to $|1\rangle$ if all conditions are fulfilled.

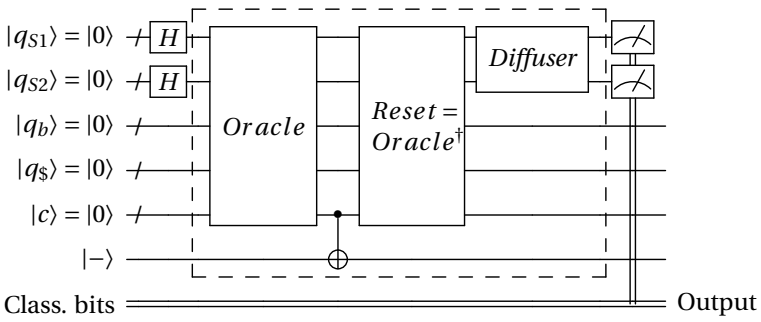
3. Grover's diffusion operator D, which multiplies the amplitude of the $|0\rangle_n$ state (or all states except $|0\rangle_n$) by -1. This has the effect of inverting all amplitudes in the quantum state about the mean, which amplifies the magnitudes of all states of interest and decreases the magnitudes of all other states.

4. These are combined into Grover's rotation operator $G = DO^{\dagger}\text{CNOT}(|-\rangle)O$.

5. The rotation operator can be repeated to amplify the state(s) of interest. A Grover's search with j rotations has j repeated applications of rotation operator G.

6. Sample the resulting state. For a problem with a solution space of size N and t valid solutions, the probability of measuring a valid solution after j rotations is $P = t \cdot |k_j|^2$, where $k_j = \frac{1}{\sqrt{t}} \sin\left((2j+1)\cdot\theta\right)$ and $\sin^2\theta = \frac{t}{N}$.

Maximum amplification of the states of interest occurs at $j = \frac{\pi}{4}\sqrt{\frac{N}{t}}$, where N is the size of the solution space, and t is the number of valid solutions.



Circuit 5.1: Circuit for Grover's algorithm as used in QISS. Within the dashed line is Grover's operator G, which can be repeated to maximize the amplification of the target states.

5.3. SIMPLIFIED MODEL FOR SHIFT SCHEDULING

The production process of vehicles is a series of consecutive steps, some of which are parallelized to increase productivity. In Figure 5.1, a sequence of five steps is depicted: a *press shop*, *body shops*, a *paint shop*, *mounting*, and two *assemblies*. Here, a singular *press shop* supplies parts to three concurrent *body shops* that assemble the vehicle's body. These assemblies then progress to a unified *paint shop* for coloring, followed by *mounting* where the vehicle's body is joined to the chassis (also referred to "marriage"). The production culminates in the *assembly*, bifurcated into two parallel processes in this example. It is noteworthy that vehicles are buffered between certain stages to modulate throughput.

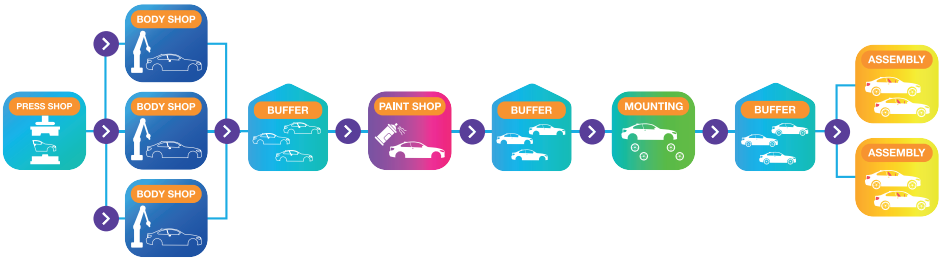


Figure 5.1: An example of an automotive productive line with five steps: a press shop, three parallel body shops, a paint shop, mounting and two separate assemblies. There are buffers between the body, paint, mounting, and assembly steps.

Figure 5.2: The structure of the simplified model: a body shop and a paint shop that share a buffer.

5.3.1. FEATURES AND CONSTRAINTS OF THE SIMPLIFIED MODEL

This chapter will consider a simplified model of the automotive production line consisting of just two shops, a body shop and a paint shop, with a single shared buffer between them. The goal is to find the optimal working time for each shop to meet an (annual) production volume target with minimal costs. The model is shown schematically in Figure 5.2.

QISS outputs a schedule with a chosen shift length for each shop for each day. A solution is *valid* if the schedule does not violate any constraints.The objective is then to find the *cheapest* valid solution, where the sum of the operating costs of the shops for the chosen shift lengths is as low as possible. Shifts are not assigned for individual employees, but at the level of shops: how many hours should the whole shop operate on a given day.

The simplified model has the following features and constraints:

- 2 shops: A body shop (S1) and a paint shop (S2).
- Both shops produce 1 unit/hour at $1/hour.
- Maximum of 1 shift per day per shop.
- Allowed shift lengths (in hours) for the body shop are [0, 5, 8, 10] and for the paint shop they are [0, 4, 7, 9].
- No additional costs for different shift assignments on consecutive days.
- The shared buffer has an initial content of 5 units.
- The buffer cannot hold less than 0 or more than 10 units at the end of each day.

- Target output volume $V^* = 8n$ for n days.
- Over- or underproduction of $\Delta = 0.05V^*$ is allowed: actual output volume is $V = 8n \pm \Delta$.

The parameters of our simplified model have been chosen to ensure the problem cannot be solved too trivially or easily. Scenarios in which the problem may become simple to solve include those where: the possible shift lengths for shops S1 and S2 are identical; the target output volume is so high (low) that the shops must essentially work the maximum (minimum) number of hours; the buffer capacity is large enough to accommodate multiple days' worth of output from shop S1. We further remark that the conditions we have chosen are realistic, e.g. it would not be practical or cost-effective to construct a buffer capable of storing multiple days' worth of output stock, which could run into thousands of vehicles for real-world scenarios.

A mathematical formulation of our objective function to minimize is given by

$$C_f = \sum_{i=1}^{\#shifts} \sum_{j=1}^{\#shops} S_{ij} C_j, \tag{5.1}$$

where *#shifts* is the product of the total number of days and the number of shifts per day, *#shops* is the total number of shops, S_{ij} denotes the assigned shift hours for shift i for shop j and C_j denotes the cost of operation per hour for shop j. The buffer is assumed to have a constant cost and does not contribute to the cost function.

For this simplified model, the number of possible schedules grows with 16^n for n days, and a year with 280 working days has 2^{1120} possible schedules. Checking all of these schedules for validity and finding the cheapest of all the valid solutions is a computationally prohibitive task. In Section 5.3.4 we provide supplementary information on the simplified model, and discuss some of the complexities that arise when considering more sophisticated, real-world models.

5.3.2. STRUCTURE OF SOLUTION SPACE

To discover methods that more efficiently explore the solution space than a pure brute-force search, an important question is whether feasible solutions (those that satisfy all the constraints) exhibit any discernible structure. If they do, then one can potentially (partially) exclude infeasible regions from the search.

In the industrial shift scheduling problem, we note that the output production volume constraint can in principle be used in this way. We have both a lower and upper total number of units that can be produced, i. e., $V_- = 8n - \Delta$ and $V_+ = 8n + \Delta$, respectively. We can easily convert these into the required number of hours that must be worked in total by each of the two shops. For example, to achieve an output of V_-, at a minimum, shop 1 must work $V_- - B_{init}$ hours, and shop 2 must work V_- hours. A similar condition can be derived to ensure that no more than V_+ units are produced.

These conditions on the shop working hours could then be used to avoid searching through solutions where the production volume is outside the allowed range. For instance, a classical brute force algorithm could be restricted in its scope by only checking solutions satisfying the above conditions. In the case of Grover's algorithm, in principle,

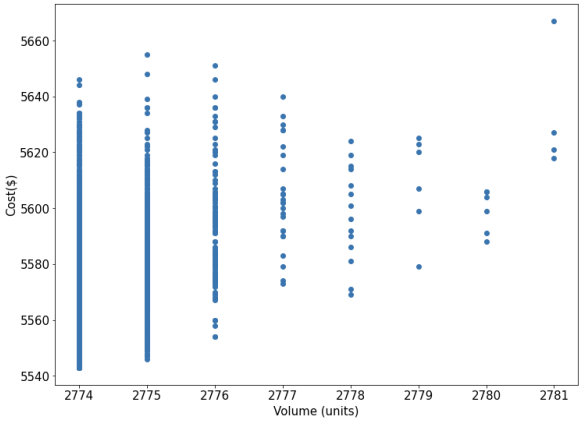


Figure 5.3: Total cost vs. output volume for 1623 feasible solutions (each represented by a blue dot) of the $n = 365$ days simplified shift scheduling model described in Section 5.3. The feasible solutions were obtained using simulated annealing.

one could prepare (e. g., using a quantum random access memory [67]) the initial state as a superposition only of solutions that respect the conditions. While the overall complexity of the problem is unchanged, in practice, the restriction to a subspace could lead to a significant speedup over a full brute-force search. Nevertheless, an exhaustive search within the identified subspace would still be necessary to obtain an exact solution.

To illustrate the nature of the feasible solution space for the simplified model for the case of $n = 365$ days, Figure 5.3 shows the relationship between the total cost and the total output volume for a total of 1623 feasible solutions found using a simulated annealing algorithm. A notable feature is the large variance in the cost of solutions that achieve the same output volume. For example, of the solutions achieving an output of 2774 units, there is a difference of around $100 in the cost of the solutions found. This shows that cost minimization is not necessarily achieved simply by limiting production.

5.3.3. A LOWER BOUND FOR THE COST

A simple way to compute a lower bound on the cost for the simplified model follows directly from considering the volume target. At the very least, we know that any feasible solution must result in an output of $V^* - \Delta$ vehicles. Therefore, we know with certainty that we must incur a minimum operating cost that corresponds to the labor required to produce those vehicles.

Denoting our lower bound for the cost value as C_{LB}, we can compute it simply as the sum of the cost of the work required by both shops:

$$C_{LB} = \left(V^* - \Delta - B_{init}\right) + \left(V^* - \Delta\right) \tag{5.2}$$

For the data specified in Section 5.3.1, we find that $C_{LB} = \$5543$.

Such a formula is easily generalizable to the case of multiple shops and buffers, multiple vehicle types, and different hourly costs and hourly production rates for the shops.

It may be possible to find a tighter lower bound than this, a question that we leave for future work.

Note that this method of obtaining a lower bound does not yield an explicit solution (shift configuration) corresponding to the cost C_{LB}. Moreover, we emphasize that C_{LB} is not necessarily the cost of the (unknown) optimal solution, which we denote C_{OPT}, although it is certainly the case that $C_{LB} \leq C_{OPT}$. Coincidentally, as we see from Figure 5.3, for the simplified model defined in Section 5.3.1, we do indeed have $C_{OPT} = C_{LB}$. However, this may not hold if the data specifying the model were to be modified, and it is also unlikely to hold for more complex versions of the industrial shift scheduling problem.

5.3.4. BEYOND THE SIMPLIFIED MODEL

More complicated versions of the industrial shift scheduling problem can consist of multiple shops, multiple buffers, multiple vehicle types, and non-linear production line structures, as depicted in Figure 5.1. One complication in tackling these problems is the increased size of the solution space, but there are a number of additional issues that can complicate the structure of the solution space itself.

For a given shift configuration, for any given shift, each shop is scheduled to work a certain number of hours, which in turn determines how many units can be processed during that shift. In the simplified model, the shop can be idle if the upstream (i.e. preceding) buffer is emptied during a shift, but a schedule is marked invalid if a chosen shift length results in buffer overflow of the downstream (i.e. following) buffer. This situation can also be handled with idle time if a shop would otherwise produce more units than the downstream buffer can hold. For example: if the downstream buffer of a shop can only hold five more units before being full, the shop can still be scheduled to run for more than five hours without causing the schedule to be marked invalid. The shop will then be idle after it has produced five units. As a result, the total output volume can exhibit a complex relationship with the number of hours worked by the shops.

In the case where there are multiple shops and intermediate buffers, the calculation of the total output volume and buffer occupation becomes yet more complicated. For instance, on a given day the occupancy of the final buffer in a production line depends on the occupancy of all upstream buffers on the previous day, and on the hours assigned to the upstream shops on the current day. Not only does this imply a larger computational runtime, but it also further complicates the relationship between the total output volume and the number of hours worked, and hence the total operating cost.

A second example is that of shared buffers, where, for instance, the output of two body shops is passed into a single buffer. The two body shops produce different vehicle models, each of which has its own total output volume target. If a single paint shop is responsible for work on both of these vehicle models, then we must specify the quantities of the different models to be passed from the buffer to the paint shop. As an example, one could choose a ratio based on the respective output volume targets. However, such a simple rule could lead to inefficiencies, since it does not take account of the actual quantities of each model in the buffer at a given time, and as a result idle time may be introduced. One could introduce more complex rules for extracting vehicles from the buffer, at the expense of a larger computational runtime.

Complications such as the two mentioned here, which are a result of more elaborate

production lines, may affect the scope for restricting the search space in the spirit of the example given in Section 5.3.2 above.

5.4. ALGORITHM DESIGN AND VALIDATION

QISS leverages Grover's algorithm [28] in combination with an adaptive search procedure, collectively referred to as Grover's adaptive search (GAS) [31, 66], to provide exact solutions to the industrial shift scheduling problem. GAS is described in Algorithm 1.

QISS returns shift schedules that satisfy the problem constraints, with the lowest cost. Compared to a classical brute-force search, QISS benefits from the asymptotic quadratic speedup delivered by Grover's algorithm. As we remarked in the introduction, this speedup may be useful in extending the scope for benchmarking the performance of heuristic algorithms designed to tackle industrial-size problem instances.

We use a simplified model of a vehicle assembly line for this proof-of-concept of QISS (Section 5.3). Based on the constraints of the model, we construct an oracle that marks input states (schedules) as valid or invalid. First, we map all possible combinations of possible shift lengths to binary numbers in equal superposition on qubit registers $|q_{S1}\rangle$ and $|q_{S2}\rangle$. This superposition of states defines the solution space, and the qubits in this register are the only qubits that are measured at the end (Section 5.4.1).

Then we define qubit registers to calculate the buffer content, total cost and solution validity (Section 5.4.2). Calculating buffer content and cost is done through adders in the Fourier basis, using the technique from [56]. Quantum Fourier Transforms are used to transform the qubits from the Fourier basis to the computational basis and back (Section 5.4.3).

Conditions are checked by addition or subtraction of relevant upper or lower limit values, so a condition is (not) met if the qubits hold a (positive) negative number. This can be easily checked because we use Two's complement encoding (Section 5.4.4).

To mark the states we want to amplify, we use a register $|c\rangle$ with a condition qubit for each of the conditions of the model. The corresponding qubit is set to $|1\rangle$ when a condition is met. Using a multi-controlled NOT gate, a single output qubit is flipped if all condition qubits are in $|1\rangle$. When the output qubit is initialized to the $|-\rangle$ state, this operation applies a phase shift to all the states that we want to mark. We verify the completed circuit for the first day (Section 5.4.5), and then extend it for scheduling multiple days (Section 5.4.6).

By updating the maximum allowed cost of the solution, only those schedules with lower cost are amplified by Grover's algorithm. When there are no more valid solutions left, the optimal solution has been found(Section 5.4.7). For bigger problem sizes, we do not know the number of valid solutions nor do we know if we have actually reached the optimal solution. This is why we need a general stop condition.

With GAS, the maximum cost at each iteration is set to the lowest cost that has been found, up to that point. We will stop the algorithm when the total number of rotations exceeds $\sqrt{N}$. This yields goods results, which we have validated using simulations for shift schedules up to three days (Section 5.4.8).

Algorithm 1: Grover's adaptive search

Goal: Find the lowest value for $y = f(x)$
Input: Oracle that flags all states x where $f(x) < y_{min}$
Set m to 1 and λ to 6/5 or any value $1 < \lambda \leq 4/3$
Set x_{best} to an initial (valid) state
Set y_{min} to a best guess for the solution
repeat
    Choose j randomly from all integers $0 \leq j < m$
    Apply Grover's search with j rotations and measure the circuit
    Set x to the measurement result
    if $f(x) < y_{min}$ **then**
        Set y_{min} to $f(x)$ and x_{best} to x
        Set m to 1
    else
        Set m to the smaller of $\lambda \cdot m$ and $\sqrt{N}$, where N equals the total number of possible solutions
until a stop condition is met
Output: x_{best} and y_{min}

5.4.1. DATA ENCODING

The shift assignments per shop function as the decision variables and are encoded in binary form, using $\log_2(i)$ qubits to store i different choices for shift assignments. Compared to a one-hot encoding, this approach uses fewer qubits and ensures that each shop is assigned a single shift length per day.

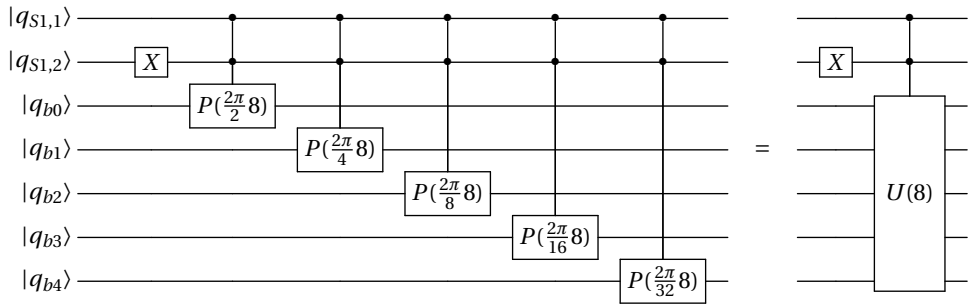
The encoding for the shift length for each shop in the simplified model is shown in Table 5.1. We need four qubits per day, two for each shop. Shift lengths are combined as $|S1\rangle|S2\rangle$, for example $|01\rangle|11\rangle$ means a shift of 5 hours for shop 1 and a shift of 9 hours for shop 2. Encoding for multiple days is done by simply adding four qubits per day to allow for solutions with different shift lengths for different days. The combination of these states for both shops and all days will make up the solution space for QISS.

Table 5.1: Encoding of shift length in hours into qubit states for shops 1 and 2.

Qubit state	Shift length in hours	
	Shop 1	Shop 2
$\|00\rangle$	0	0
$\|01\rangle$	5	4
$\|10\rangle$	8	7
$\|11\rangle$	10	9

The number of units in the buffer is stored in a separate register in Two's complement form. Compared to other signed number representations, Two's complement has the advantage that the operations for addition, subtraction, and multiplication are identical to those for unsigned binary numbers. With Two's complement, the most significant bit

(MSB) of a binary number is '0' for positive numbers and zero, and '1' for negative numbers, which simplifies checking for the constraints later. To convert a negative number to its b-bit Two's complement equivalent, we can add 2^b and convert the resulting number as if it were an unsigned binary number. For example, the number -2 in three-bit Two's complement is $2^3 - 2 = 8 - 2 = 6 =$'110'.



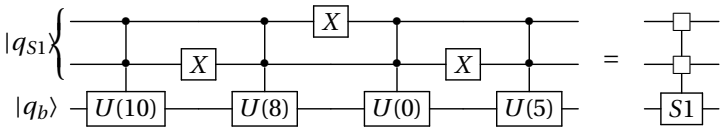
Circuit 5.2: Conditional data encoding of the number 8 on qubit vector $|q_b\rangle = |q_{b0}q_{b1}q_{b2}q_{b3}q_{b4}\rangle$. In future circuits, we will refer to this type of circuit as the controlled gate $U(a)$ on the right for the encoding of the number a.

We use circuits like the one in Circ. 5.2 [56] to encode the number of units produced per shift. This circuit encodes the decimal value 8 into the buffer qubit register $|q_b\rangle$ when the decision variable register $|q_{S1}\rangle$ is in state $|10\rangle$. This corresponds to the 8 units produced during the 8-hour shift of Shop 1.

The phase gates $P(\lambda)$ in this circuit apply a rotation around the Z-axis by angle λ to the qubit. If the phase gate is controlled by one or multiple other qubits, it only applies the rotation when all controlling qubits are $|1\rangle$. The matrix representation for this gate is:

$$P(\lambda) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{bmatrix} \tag{5.3}$$

This can be used to encode the number of units produced for each possible shift length, which is shown in Circ. 5.3.



Circuit 5.3: Encoding the number of units produced for the body shop (S1) in superposition onto $|q_b\rangle$, using the U(a) gate from Circ. 5.2.

5.4.2. QUBIT REGISTERS

QISS uses six distinct qubit registers, which are the following:

Table 5.2: Description of the qubit registers used in QISS, including the number of qubits in each.

Register	Description	Qubits: 1 day	n days
$\|q_{S1}\rangle$	Shop 1	2	$2n$
$\|q_{S2}\rangle$	Shop 2	2	$2n$
$\|q_b\rangle$	Buffer	5	6
$\|a\rangle$	Ancilla	5	$6n$
$\|q_\$\rangle$	Cost	6	$\lceil \log_2(19n) \rceil + 1$
$\|c\rangle$	Conditions	4	$n+3$
$\|-\rangle$	Mark states	1	1
	Total	25	$11n + 11 + \lceil \log_2(19n) \rceil$

1. **Decision variables** $|q_{S1}\rangle$ and $|q_{S2}\rangle$: This register stores the assigned shifts to each shop. For i different possible shift assignments per shop, we need $\lceil \log_2(i) \rceil$ qubits. Each shop has four possible choices of shift assignment, so we need $\lceil \log_2(4) \rceil = 2$ qubits per shop per day. For n days, we will need $\lceil n \cdot \log_2(i) \rceil$ qubits.

2. **Buffer occupancy** $|q_b\rangle$: This register stores the number of units in the buffer after a given day's work. It must contain enough qubits to capture buffer overflows, which correspond to infeasible solutions. For the simplified model, this is the case when the buffer is already filled to the maximum allowed value B_{max}, and the next day, shop 1 produces the maximum number of units while shop 2 does not consume any units. This leads to a buffer occupancy of $10 + 10 = 20$ units, which requires $\lceil \log_2(10 + 10) \rceil + 1 = 6$ qubits. This is the maximum buffer occupancy that we need to calculate accurately, since there is no reason to have accurate computations of buffer occupancy for subsequent days after the solution has already been marked infeasible.

3. **Ancilla qubits** $|a\rangle$: This register is used to impose a floor of zero on the buffer occupancy, since it cannot contain a negative number of vehicles. For every day that needs to be scheduled, the $MAX(0, \tilde{B})$ operation described in Section 5.4.3 requires as many ancilla qubits as there are qubits in the buffer register, which is $6n$ for n days for the simplified model. On the first day, only 5 qubits are required, because the buffer value cannot exceed $5+10 = 15$ units. This can be taken advantage of by using a modified $MAX(0, \tilde{B})$ circuit.

4. **Cost qubits** $|q_\$\rangle$: This register records the operational cost of implementing a shift schedule. It must contain enough qubits to store the maximum possible cost without overflowing. This can be done as an unsigned number because no negative costs are possible. The maximum cost per day for the simplified model is \$19, so for n days we need a cost register of $\lceil \log_2(19n) \rceil + 1$ qubits.

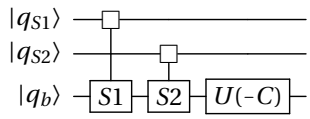
5. **Condition qubits** $|c\rangle$: This register keeps track of all the different constraints, and whether they are satisfied or violated. For each constraint, we thus require one qubit to store a boolean value. Three qubits are needed for one day, and we need $n + 2$ qubits for n days.

6. **Marking of valid states** $|-\rangle$: A single qubit that is initialized in the state $|-\rangle$. By applying a multi-controlled not gate from the condition qubits to this output qubit, we apply a phase of -1 to all valid solutions, as indicated by all the constraints being met. The subsequent operations of Grover's algorithm will amplify the amplitude of these marked valid solutions.

An overview of the number of qubits used by the oracle can also be found in Table 5.2. In total, QISS uses 25 qubits to find the optimal schedule for a single day. For n days the required number of qubits is $11n+11+\lceil \log_2(19n) \rceil$, or $11n+10+\lceil \log_2(19n) \rceil$ when there is a separate implementation of a modified $MAX(0,\tilde{B})$ for the first day.

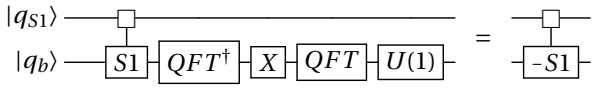
5.4.3. CALCULATING BUFFER VALUES

We will need to perform addition and subtraction to calculate the value of the buffer occupancy on each day, and the total output volume produced by the shops. We can apply the relevant SX or $U(X)$ gates to the target qubits, as shown in Circ. 5.4.



Circuit 5.4: Calculating $S1+S2-C$

Rather than defining a separate subtraction method, we negate the value to be subtracted, and subsequently use the addition method described in Section 5.4.1. The negative of the value A is calculated as $(-1)*A = 2^m - A$ for a register with m (qu)bits using Two's complement. For values already encoded in the quantum circuit, the negation of the bits is done by applying an X-gate to all the qubits in the computation basis, and then adding 1. This circuit is shown in Circ. 5.5.

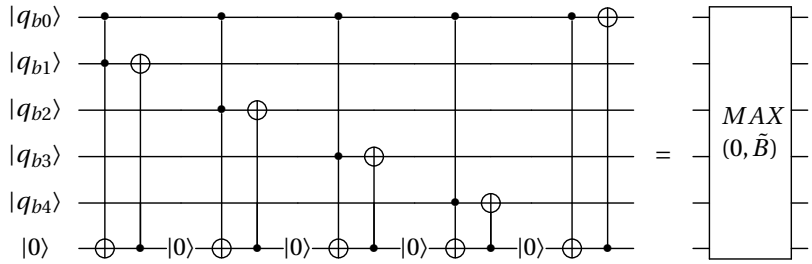


Circuit 5.5: Calculating $|q_b\rangle = -S1$

Before constructing the complete circuit, we need to define a final building block. This is the $MAX(0,\tilde{B})$ operation, which enforces that the buffer always holds zero or more units. If the scheduled shop hours lead to less than zero units in the buffer, the number of units in the buffer is instead set to zero. The circuit to calculate $MAX(0,\tilde{B})$ is shown in Circ. 5.6.

After calculating the buffer occupancy, we must transform the register back to the computational basis for the application of the $MAX(0,\tilde{B})$ operation and the constraint checks. Transformations from the computational to the Fourier basis is done with Quantum Fourier Transforms (QFT), and to return to the computational basis an inverse QFT is used.

As we are encoding the buffer occupancy value in Two's complement, we can use the MSB ($|q_{b0}\rangle$) to check if the number is smaller than zero. If that is the case, we set all the other qubits to zero. We do this for each qubit in turn, by setting an ancilla qubit to $|1\rangle$ if both the MSB and the current target bit are in state $|1\rangle$ with a Toffoli gate. Then, we apply a CNOT from an ancilla in the $|0\rangle$ state to the current target bit, setting it to zero. The inverse oracle (Oracle†) is also used in the circuit (see Circ. 5.1), so a "fresh" ancilla needs to be used for each bit of the buffer register. The circuit requires b qubits per day for the b qubits in buffer register $|q_b\rangle$.

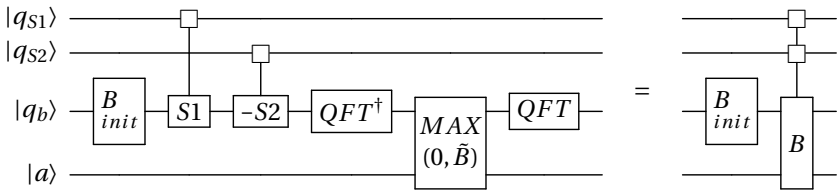


Circuit 5.6: Circuit for forcing the value of $|q_b\rangle$ to be greater than or equal to zero, when $|q_b\rangle$ is stored in Two's complement. The $|0\rangle$ in the circuit means that the ancilla qubit is reset to zero, or a "fresh" ancilla in the $|0\rangle$ is used. The reset operation involves non-reversible measurement operations, but the circuit can be made fully reversible with the additional ancilla qubits. This circuit will be referred to as $MAX(0, \tilde{B})$.

The final value for the buffer B_{out} after each day can be calculated as:

$$\tilde{B} = B_{init} + S1 - S2$$
$$B_{out} = MAX(0, \tilde{B})$$

This circuit is shown in Circ. 5.7. The value of $-S2$ can be implemented from $S2$ as in Circ. 5.5, however, since $S2$ is generated from the classical input to the circuit, it can also be implemented without using additional gates. This is done by replacing all the shift lengths (s_i) for S2 with their corresponding $2^m - s_i$ and encoding the circuit $2^m - S2$ with these new values as in Circ. 5.3. Because we are using Two's complement, this is equivalent to using $-S2$. Directly negating the input values uses fewer gates than the method in Circ. 5.5, and is what we will use for the rest of the circuits.



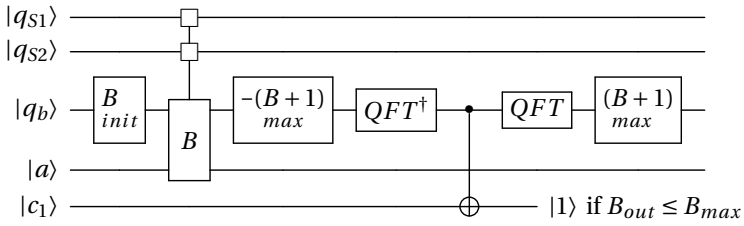
Circuit 5.7: Calculation of buffer value B_{out} for the simplified model after a single day

5.4.4. CONSTRAINT CHECKING

After each day of operation, the buffer content must not exceed its maximum capacity B_{max}. We use this as a condition within Grover's algorithm, to distinguish feasible and infeasible solutions, by introducing a qubit $|c_1\rangle$ that will be $|1\rangle$ when the condition is met, and $|0\rangle$ otherwise.

$$c_1 = \begin{cases} 1 \text{ if} B_{out} \leq B_{max} \Rightarrow B_{out} - (B_{max}+1) < 0 \\ 0 \text{ if } B_{out} > B_{max} \end{cases} \tag{5.4}$$

In order to set the value of $|c_1\rangle$, we first calculate $B_{out} - (B_{max}+1)$, and subsequently apply an inverse quantum Fourier transform, $QFT^\dagger$, to return the buffer qubit register to the computational basis. If the condition is met, then the buffer register state then corresponds to a negative (binary) number, and we can apply a CNOT from the MSB to our condition qubit $|c_1\rangle$. Finally, we need to restore the buffer register to its state before this constraint check, by applying a QFT to return to the Fourier basis, and then adding $(B_{max}+1)$. The circuit for this operation is shown in Circ. 5.8, with the buffer register at the output again holding a value of B_{out}.

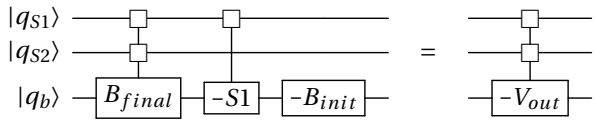


Circuit 5.8: Evaluation of maximum buffer capacity condition $B_{out} \leq B_{max}$, with B_{init} and B as defined in Circ. 5.7. The CNOT is controlled by the MSB of the register $|q_b\rangle$. The buffer register is returned to its original state B_{out} through a quantum Fourier transform and by adding the value of $B_{max}+1$.

The output volume V_{out} of the simplified model is calculated as:

$$V_{out} = B_{init} + S1 - B_{final}$$

where B_{final} is the content of the buffer at the end of the day. The circuit will (initially) be applied only for 1 day, in which case $B_{final} = B_{out}$. To avoid the additional gates required to calculate $-B_{final}$ (Circ. 5.5), we can instead calculate $-V_{out} = -B_{init} - S1 + B_{final}$. We can calculate the output volume on the same qubits that are used to store the buffer occupancy value, which requires fewer qubits than using a separate qubit register and saves copying or recomputing the data in the buffer. This means that the volume requirement cannot easily be checked at the end of each day, but this is not required by the constraints of the simplified model. $-S1$ and $-B_{init}$ can be easily calculated using the Two's complement method as before. Because we are computing $-V_{out}$, the constraints for the output volume are defined accordingly. We call these output volume constraints c_2 and c_3, with values as in Equations (5.5) and (5.6).



Circuit 5.9: $-V_{out} = -B_{init} - S1 + B_{final}$

$$c_2 = \begin{cases} 1 \text{ if } V_{out} \geq V_{low} \Rightarrow -V_{out} + V^* - \Delta \leq 0 \\ 0 \text{ if } V_{out} < V_{low} \Rightarrow -V_{out} > -V_{low} \end{cases} \tag{5.5}$$

$$c_3 = \begin{cases} 1 \text{ if } V_{out} \leq V_{up} \Rightarrow -V_{out} + V^* + \Delta \geq 0 \\ 0 \text{ if } V_{out} > V_{up} \Rightarrow -V_{out} < -V_{low} \end{cases} \tag{5.6}$$

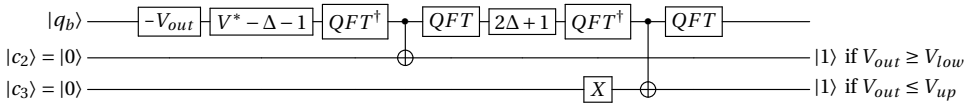
The lower limit for the tolerance is $V_{low} = V^* - \Delta$, for target volume V^* and tolerance Δ. The upper limit for the tolerance is defined as $V_{up} = V^* + \Delta$.

To set qubit c_2, we need to transform the inequality from Equation (5.5) to $-V_{out} + (V^* - \Delta - 1) < 0$ (where '1' is the minimum difference between numbers), which can be calculated on the buffer register. Then we use a CNOT from the MSB of the buffer register to qubit c_2 to flip the qubit to $|1\rangle$ when the condition is met, and keep it at $|0\rangle$ otherwise.

For qubit c_3, we can do the same. We calculate $-V_{out} + (V^* + \Delta)$ and then set c_3 based on the MSB of the result. An X-gate is needed to flip c_3, since the condition is met when the MSB is in state $|0\rangle$ (when the result of the calculation is bigger than or equal to zero).

To avoid recomputing V_{out}, we can compute the result for evaluating c_3 from the result for c_2, simply by adding $(2\Delta + 1)$ to the latter. This works because we are first comparing to the lower limit, and the output volume should be within 2Δ of it to be within the upper limit. The addition of '1' is again the minimum precision.

Combining these will give the circuit in Circ. 5.10.

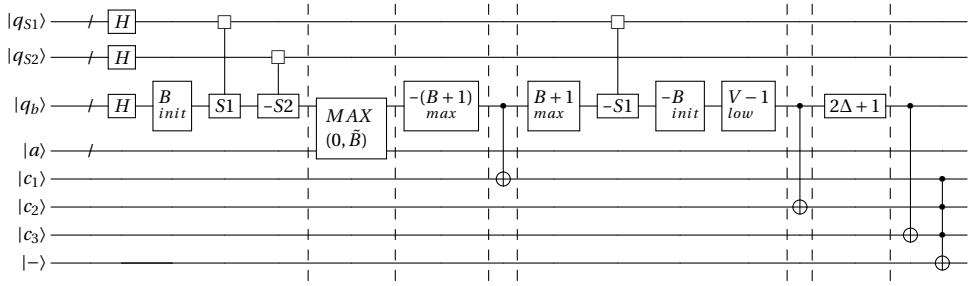


Circuit 5.10: Circuit for evaluating the output volume conditions $|c_2\rangle$ and $|c_3\rangle$.

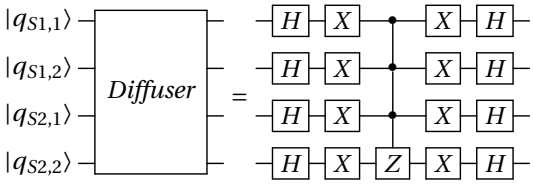
5.4.5. COMPLETED CIRCUIT FOR DAY ONE

The complete circuit for scheduling one day is shown in Circ. 5.11. This circuit can be used as the oracle in the circuit in Circ. 5.1, where the sub-circuit within the dashed box is applied a specific number of times, namely the number of Grover rotations. The diffuser operator in Circ. 5.1 is implemented as Circ. 5.12.

All possible combinations of shifts for one day are shown in Table 5.3. For each possibility, the buffer content and output volume have been calculated, as well as the expected value of the condition checks.



Circuit 5.11: The complete circuit for the simplified model with buffer and volume constraints for scheduling of a single day. The $QFT^{\dagger}$ and QFT operations have been abstracted as vertical dashed lines. Every time a dashed line is crossed, the buffer is converted from the Fourier basis to the computational basis or back. This circuit is used as the oracle in Grover's search algorithm.



Circuit 5.12: Diffuser for Grover's algorithm

The schedules where shop 1 has a long shift and shop 2 has a short shift are not valid, because, at the end of the day there will be more units in the buffer than its maximum capacity. The target volume of 8 ± 0.4 units per day cannot be met when scheduling for a single day, since shop 2 can only produce 7 or 9 units in a shift. If we relax the constraint (set Δ to 1), the production volume never exceeds the maximum. The minimum production target is met if shop 2 has a shift of at least 7 hours. There are 16 possible schedules for the simplified model, of which six meet all the requirements.

To validate that our approach gives the expected results,Circ. 5.11 was implemented in Qiskit and simulated with Qiskit Aer, assuming noiseless, fully-connected qubits. The results from 1000 measurements of the output state of the circuit are shown in Figure 5.4, with the six valid solutions clearly amplified.

5.4.6. EXTENDING TO MULTIPLE DAYS

To extend the oracle to multiple days, we need to implement several changes compared to the circuit for a single day. Firstly, to encode the shifts for each shop for all possible days, we will need $2n$ qubits per shop for n days.

Secondly, we need to calculate the buffer occupancy values and check the buffer constraints after each day. The buffer value after one day is $B_1 = MAX(0, \tilde{B}_1)$, where $\tilde{B}_1 = B_{init} + S1_1 - S2_1$. Combined, this gives $B_1 = MAX(0, B_{init} + S1_1 - S2_1)$. The buffer value after two days will be $B_2 = MAX(0, B_1 + S1_2 - S2_2)$, and after n days it will be:

$$B_n = MAX(0, B_{n-1} + S1_n - S2_n)$$

Table 5.3: Expected output for the simplified model for one day, bolded rows are solutions that satisfy all the constraints (c_1: $B_{out} \leq B_{max}$, c_2: $V_{out} \geq V_{low}$, c_3: $V_{out} \leq V_{up}$)

$\|S1\rangle$	$\|S2\rangle$	B_{init}	$U(S1)$	$U(S2)$	B_{out}	V_{out}	$Cost$	c_1	c_2	c_3
00	00	5	0	0	5	0	\$0	1	0	1
00	01	5	0	4	1	4	\$4	1	0	1
00	10	5	0	7	0	5	\$7	1	0	1
00	11	5	0	9	0	5	\$9	1	0	1
01	00	5	5	0	10	0	\$5	1	0	1
01	01	5	5	4	6	4	\$9	1	0	1
01	**10**	**5**	**5**	**7**	**3**	**7**	**\$12**	**1**	**1**	**1**
01	**11**	**5**	**5**	**9**	**1**	**9**	**\$14**	**1**	**1**	**1**
10	00	5	8	0	13	0	\$8	0	0	1
10	01	5	8	4	9	4	\$12	1	0	1
10	**10**	**5**	**8**	**7**	**6**	**7**	**\$15**	**1**	**1**	**1**
10	**11**	**5**	**8**	**9**	**4**	**9**	**\$17**	**1**	**1**	**1**
11	00	5	10	0	15	0	\$10	0	0	1
11	01	5	10	4	11	4	\$14	0	0	1
11	**10**	**5**	**10**	**7**	**8**	**7**	**\$17**	**1**	**1**	**1**
11	**11**	**5**	**10**	**9**	**6**	**9**	**\$19**	**1**	**1**	**1**

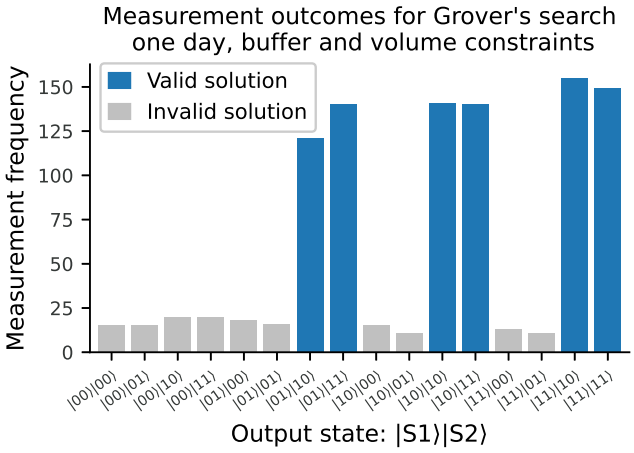
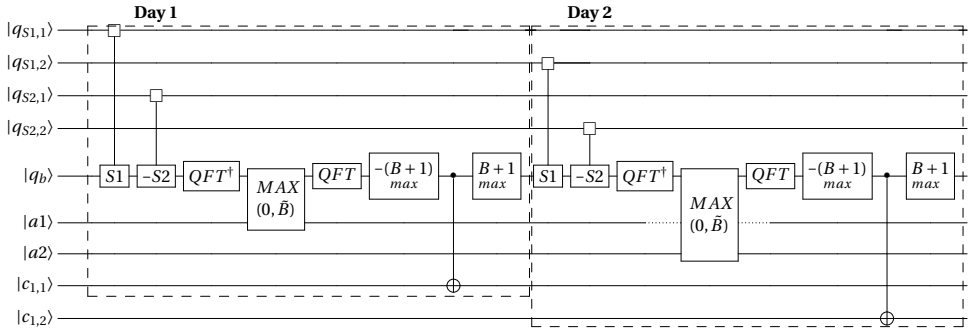


Figure 5.4: Simulated measurement outcomes using Qiskit for the implementation of Grover's search for scheduling one day and only buffer and volume constraints. States corresponding to valid solutions are shown in dark blue.

For each day, the values produced by each shop are added or subtracted from the buffer register, controlled by the qubits in the corresponding shop register. Then the value for $MAX(0,\tilde{B})$ need to be calculated, which requires a new set of ancilla qubits ($|a_n\rangle$) per day to keep the circuit reversible. Finally, a conditional qubit ($|c_{1,n}\rangle$) is set depending on if the buffer exceeds the maximum allowed buffer content.

This means that we repeat everything in Circ. 5.7 after the gate for B_{init}, with separate controls and ancillas for each day. For two days, this looks like the circuit in Circ. 5.13. After n repeats for n days, the output volume can be calculated. This can be done using the same circuit as for one day (Circ. 5.9), but instead of subtracting the production of S1 once, it is done n times for n days.



Circuit 5.13: Calculation of buffer value B_{out} for the simplified model for 2 days, including calculation of $MAX(0,\tilde{B})$ for each day. Conditional qubits $|c_{1,1}\rangle$ and $|c_{1,2}\rangle$ are set depending on if the buffer exceeds the maximum allowed buffer content for both days.

Checking whether the output volume falls within the allowed limits is similar to the case of a single day, since it only needs to be checked at the end. The only differences are the target output volume V^* and the allowable margin Δ, which need to be adjusted depending on the number of days that are being simulated.

For verification, we implemented Circ. 5.13 (corresponding to two days of operation) in Qiskit and simulated 1000 circuit shots using Qiskit Aer, which produced the measurement results shown in Figure 5.5. The 22 valid solutions (shown in blue) are clearly amplified over the 244 non-valid solutions.

5.4.7. COST CONSTRAINT

In the framework of Grover adaptive search, the objective function corresponding to the total factory operating cost can be handled as an additional constraint by setting a certain maximum allowed cost. To store this total cost, we introduce an additional qubit register, which must be large enough to avoid numerical overflow when both shops are operated for the maximum number of hours daily. We also add a condition qubit for the cost constraint.

Computing the total cost is straightforward: we add the cost corresponding to each shift to the register with the same controlled adders as we did for the shifts. The same set of shop qubits also controls them. In the case of the simplified model, the cost and the number of produced units are the same, but this is not necessarily the case for more realistic models.

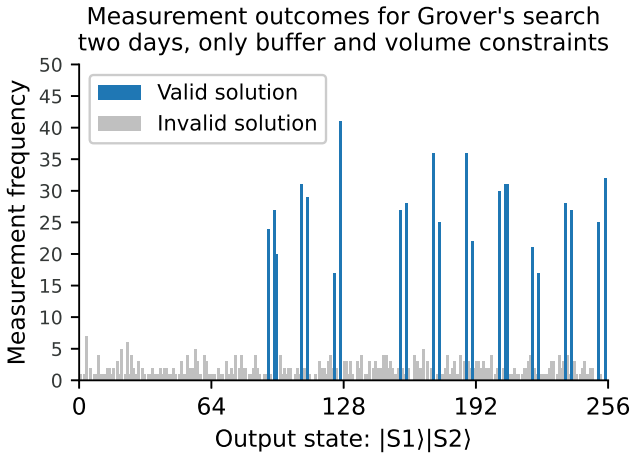
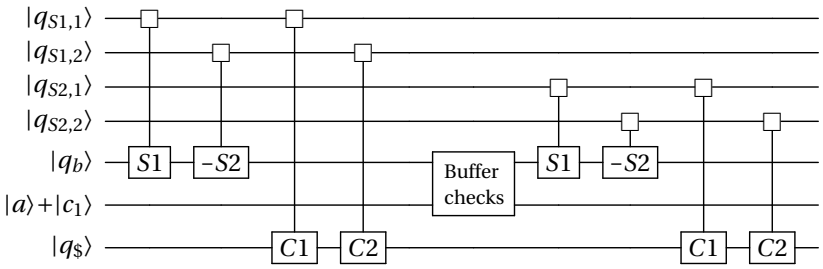


Figure 5.5: Simulated measurement outcomes using Qiskit for the implementation of Grover's search for scheduling two days and only buffer and volume constraints. States corresponding to valid solutions are shown in dark blue, output states are converted from binary to decimal.

Adding the cost can be done at any point in the circuit, but for simplicity's sake, we will add the costs for each shop just after the corresponding values have been added to the buffer.

At the end of the circuit, the cost constraint is checked with a similar construction as for the maximum and minimum volume constraints. To avoid adding the minimum precision, we define the cost constraint as $C_{out} < C_{max}$ where C_{out} is the calculated cost of the solution, and C_{max} is the maximum allowed cost. This means that we can calculate $C_{out} - C_{max}$ at the end of the circuit, and the condition is met if the result is less than zero, which means that the MSB is 1. We use this to set a single condition qubit with a CNOT.



Circuit 5.14: Calculation of the buffer content and the cost for shops $S1$ and $S2$ for two days. All elements are as defined before, with additionally cost register $|q_\$\rangle$, and gates $C1$ and $C2$, which are the costs per shift for shops 1 and 2.

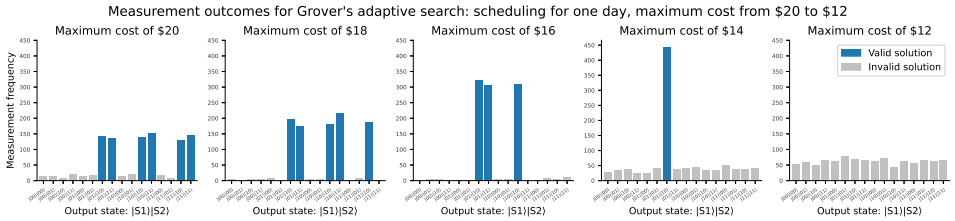


Figure 5.6: Simulated measurement outcomes for Grover's adaptive search for scheduling 1 day with manually set value for the cost constraint ($C_{out} < C_{max}$) starting at $20 and decreasing with steps of $2 until $12, at which point there are no solutions left that satisfy all the constraints.

5.4.8. VALIDATING THE STOP CONDITION OF GAS

Having constructed the oracle and verified its ability to produce expected solutions, we now proceed to investigate a proper *stop condition* for (see Algorithm 1).

There are many options for the stop condition of GAS: the loop can be stopped when no improvement has been achieved in the last few of iterations when a threshold lowest cost has been achieved, when a certain amount of time has passed, when a certain number of iterations has been performed, etc.

We will stop the GAS procedure when the combined total rotation count (number of times that Grover's rotation operator has been applied) exceeds $\sqrt{N}$, where N is the size of the search space. To validate our decision, we have simulated 10,000 runs of GAS and kept track of the number of rotations and the current best cost after each loop. The number of valid solutions and associated costs correspond to those for the shift schedules for three subsequent days. Each shop has four possible shift lengths per day, so the total number of possible schedules $N = 4^{2n}$. For three days, the total problem size is thus $N = 4^{2\cdot 3} = 4096$.

Each simulated run of GAS ran for a total of $2\sqrt{N} = 128$ rotations. For each loop of QISS, the specific number of rotations was determined randomly as outlined in Algorithm 1. The initial cost was set to $60, which is three dollars more than the cost of operating both shops, the maximum allowed number of hours for all three days. When a valid solution is found, i.e. a solution that satisfies all the constraints, the initial cost is replaced with the cost for the valid solution. This allows us to see how many rotations it took QISS to arrive at a valid solution. After each loop was completed, the running totals for the number of rotations with corresponding lowest found cost were stored. After 10,000 runs of GAS, the average, median, and percentile intervals were calculated from the results. These can be found in Figure 5.7.

As can be seen from the figure, the minimum cost found by QISS decreases exponentially with the number of Grover rotations, and the chance that we have landed on the cheapest possible shift schedule increases accordingly. After a total of $\frac{\pi}{4}\sqrt{N} = 51$ rotations, more than 95% of runs have landed on the minimum cost of 41. After $\sqrt{N}$ rotations, this increases to more than 99%. Therefore, we will set the stop condition of GAS to a number of rotations equal to $\sqrt{N}$, which yields good results while still ensuring an asymptotic quadratic speedup over a classical brute-force search.

The exact rate at which the found cost decreases is dependent on the problem size,

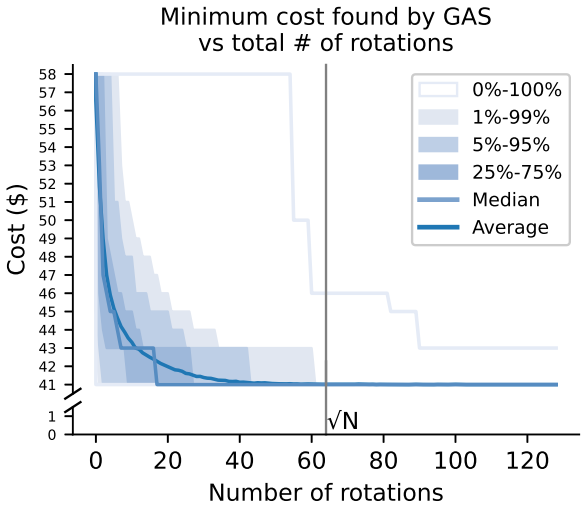


Figure 5.7: Minimum cost found by GAS after a certain number of rotations for finding shift schedules for three days, with the average, median and percentile intervals calculated from 10,000 runs of the adaptive search.

how many possible solutions are valid, the distribution of the costs associated with each valid solution, and the number of valid solutions that share the minimum cost. We do not have this data for larger problem sizes, but we expect that these results hold up for larger problem sizes.

5.5. GATE REQUIREMENTS

In this section, we show the gate requirements for each high-level operator used in QISS at different levels of gate decomposition. We use this to find the total number of elementary gates required by QISS for scheduling a single day without a cost constraint, this corresponds to the circuit at the end of Section 5.4.5.

5.5.1. HIGH-LEVEL OPERATORS

The number of elementary gates for each of the high-level operators used in QISS is shown in Table 5.4. Gates are grouped by type: all types of 1-qubit gates, multi-controlled Pauli Z or X gates, double-controlled phase gates, (single) controlled phase gates, double-controlled X (Toffoli) gates, and controlled X gates. The components correspond to the components outlined before.

Some of the components require 3-qubit gates, which can be decomposed into 1- and 2-qubit gates. The controlled U(X), the S(X) are both implemented with CCPhase gates, which can be decomposed into 3 CPhase and 2 CNOT gates each. The $MAX(0, \tilde{B})$ component is implemented with CCNOT (also called Toffoli) gates, which can be decomposed into 9 1-qubit gates and 6 CNOTs each. Gate totals for these components with these decompositions are also shown in the table. The diffuser component requires a multi-controlled Z gate, which can also be decomposed into 1- and 2-qubit gates. The

Table 5.4: Number of elementary gates required for each of the high-level operators used in QISS. The types of elementary gates here are all types of 1-qubit gates, multi-controlled Pauli Z or X gates (MCZ/MCX), double-controlled phase gates (CCPhase), (single) controlled phase gates (CPhase), double-controlled NOT (Toffoli) gates (CCNOT), and controlled NOT gates (CNOT).

Circuit component	1-qubit gates	MCZ/ MCX	CCPhase	CPhase	CCNOT	CNOT
Output init.	1					
Superposition for shops	2					
Superposition for buffer	5					
CU(X)			5			
Decomposition of CU(X)				15		10
U(X)	5					
S(X)	4		20			
Decomposition of S(X)	4			60		40
QFT	5			10		
IQFT	5			10		
$MAX(0,\tilde{B})$					4	6
Decomp. of $MAX(0,\tilde{B})$	36					30
Diffuser	16	1				

number of such gates depends on the specific decomposition method and on the number of control qubits. For this reason, the gate has been left as-is, and no decomposed gate count is provided.

5.5.2. GATE REQUIREMENTS FOR QISS

We used the totals in Table 5.4 to find the gate requirements for QISS for scheduling one day and without the cost constraint, as it is at the end of Section 5.4.5. This was used as a verification step when developing the algorithm. We have included it as a companion to the qubit requirements of QISS in Section 5.4.2. The total gate requirement can be found in Table 5.5.

The oracle for QISS is split up into its components:

- The initialization requires a single qubit (Hadamard or general initialization gate) for the shop qubits, the buffer qubits and the output qubit.
- Some addition operations could be merged, so the final circuit for the oracle for the simplified model has 4 adders: B_{init}, $-(B_{max}+1)$, the combined $(B_{max}+1)-B_{init}+(V_{low}-1)$ and $2\Delta+1$, each requiring 5 1-qubit phase gates.
- The oracle has 3 S(X) operations: $S1$, $-S2$ and $-S1$, each implemented with 4 1-qubit X-gates and 20 CCPhase gates.
- The $MAX(0,\tilde{B})$ operation is implemented with 4 CCNOTs and 6 CNOTs, with a leading Inverse QFT (IQFT) and followed by a QFT operation, each implemented with 5 1-qubit Hadamard gates and 10 CPhase gates.
- The c_1 and c_2 constraints (maximum buffer content and minimum output volume), are each implemented with a leading IQFT, a single CNOT and followed by

a QFT. In total, this is 10 1-qubit Hadamard gates, 20 CPhase and 1 CNOT gate for both conditions.

- The final c_3 condition requires only a leading IQFT, because we do not need to do any further calculations on the buffer qubits, and it therefore does not need to be "reset" as part of the oracle. In addition to the IQFT, it is implemented with a 1-qubit X gate and a CNOT, for a total of 6 1-qubit gates, 10 CPhase gates and a CNOT.
- In total, the oracle is thus implemented with 78 1-qubit gates, 60 CCPhase gates, 70 CPhase gates, 4 CCNOTs and 9 CNOTs.

Table 5.5: Total number of elementary gates required for Grover's algorithm for QISS for scheduling one day without a cost constraint, with the number of gates per component as in Table 5.4.

Circuit component	1-qubit gates	MCZ/ MCX	CCPhase	CPhase	CCNOT	CNOT
Oracle	78	0	60	70	4	9
Initialization	10					
B_{init}	5					
S1	4		20			
-S2	4		20			
IQFT	5			10		
$MAX(0,\tilde{B})$					4	6
QFT	5			10		
-(B_{max}+1)	5					
c_1	10			20		1
B_{max}-B_{init}+V^*..	5					
-S1	4		20			
c_2	10			20		1
2Δ-1	5					
c_3	6			10		1
Set output		1				
Oracle†	78		60	70	4	9
Diffuser	16	1				
Measurement	4					
Total:	176	2	120	140	8	18

The complete implementation of Grover's algorithm for the simplified model consists of the following components:

- The oracle, as above, which includes initialization.
- A multi-controlled X gate to set the output qubit based on the result of the condition qubits.
- The inverse of the oracle to reset the circuit, which requires the same number of gates as the oracle.

- Grover's diffusion operator, which consists of 2 X-gates and 2 Hadamard gates per shop qubit, and one multi-controlled Z gate over all shop qubits. This totals $2*4+2*4=16$ 1-qubit gates and 1 MCZ gate.
- And finally, a measurement on each of the shop qubits.
- In total, the implementation of Grover's algorithm for the simplified model (for a single iteration) requires 176 1-qubit gates, an MCX and an MCX gate, 120 CCPhase gates, 140 CPhase gates, 8 CCNOT gates and 18 CNOT gates.

5.6. CONCLUSION

We have introduced QISS, a quantum algorithm based on Grover's adaptive search for industrial shift scheduling problems with production target and intermediary storage constraints, a situation found in settings such as the automotive industry. We show the construction of a quantum circuit to implement the necessary Grover's oracle for an arbitrary number of days of factory operations, within the context of a simplified model comprising two shops and one buffer. For small problem instances, we have numerically corroborated the performance of the algorithm. In particular, in our examples we verify that $\sqrt{N}$ applications of Grover's rotation operator suffice to find the optimal solution, where N is the size of the solution space. This shows that an asymptotic quadratic speedup can in principle be achieved over classical unstructured search. In practice, this may be useful in at least two scenarios: (1) to obtain exact solutions to problems of small or modest size, in the context of benchmarking heuristic algorithms designed to scale to much larger, industrial problem sizes; (2) to use QISS as an integral component of a heuristic strategy, where exact solutions for short time periods are used to construct a solution for a longer time period.

Our work lays the foundation for future research in several directions. Firstly, in the context of the simplified model, QISS could be used as a target algorithm to study and benchmark the performance of quantum computing systems. Specifically, in the pre-fault-tolerant era one could test the different circuit primitives, and investigate their susceptibility to noise on different quantum computing platforms. Techniques to suppress, mitigate, or detect certain types of error may be found, which could allow for improved results, and yield useful insights for running the algorithm in the future on fully fault-tolerant machines. In the fault-tolerant era, QISS could be incorporated into application-level benchmark frameworks [60].

Secondly, we have focused on a simplified shift scheduling model with only two shops and a single buffer. It would be interesting to develop Grover's oracles for more complex, more realistic instances of industrial shift scheduling, such as the one shown in Figure 5.1. For example, new circuit primitives may need to be designed due to the intricacies resulting from multiple and/or shared buffers (see Section 5.3.4). Moreover, while the resulting circuits would very quickly become intractable to simulate on a classical computer, even for a single day, it would allow a quantitative investigation of the additional circuit complexity arising from the presence of multiple shops and buffers.

Thirdly, in this work we have developed quantum circuits at the *logical* level, which assumes qubits to be perfectly noiseless. To implement these circuits in practice on

quantum computers, which are always subject to some degree of noise, a fault-tolerant quantum error correction scheme would be necessary. Quantum error correction introduces significant resource overheads in terms of the number of required qubits and gate operations. Key quantities of interest such as the total number of *physical* qubits and the total computational runtime can be estimated through frameworks incorporating different layers of assumptions on the architecture of a fault-tolerant quantum computer, and the compilation of quantum programs [22, 154]. Subsequently, one could compare these estimates to those of classical computing approaches, and seek to identify the scale at which QISS may deliver a speedup in practice for unstructured search, taking into account the gap in execution time for basic quantum and classical circuit operations [14, 84].

Finally, we have not investigated whether the industrial shift scheduling problem is amenable to solution by dynamic programming methods, however we note that quantum algorithms capable of exploiting optimal substructure have been described in [9]. Therefore, if a speedup over unstructured search can be obtained classically with dynamic programming, it may also be possible to incorporate this into QISS.

5

ACKNOWLEDGEMENTS

For this chapter, we thank Vladislav Samoilov and Nikolas Beulich for providing the initial problem statement and for stimulating discussions, and Ewan Munro and Marvin Erdmann for their contributions to the paper and the overall project.

6

ASSESSING THE REQUIREMENTS FOR INDUSTRY RELEVANT QUANTUM COMPUTATION

In this chapter, we use open-source tools to perform quantum resource estimation to assess the requirements for industry-relevant quantum computation. Our analysis uses the problem of industrial shift scheduling in manufacturing and the quantum industrial shift scheduling algorithm. We use existing scenarios from literature for multiple qubit technologies for our initial resource estimations, and we find that superconducting qubits are the most promising for our application. Based on these findings, we do further resource estimations based on current and theoretical high-fidelity superconducting qubit platforms. We find that the execution time of gate and measurement operations determines the overall computational runtime more strongly than the system error rates. Moreover, achieving a quantum speedup would not only require low system error rates ($\mathbf{10^{-6}}$ or better), but also measurement operations with an execution time below **10** ns. This rules out the possibility of near-term quantum utility for this use case, and suggests that significant technological or algorithmic progress will be needed before quantum utility can be achieved.

This chapter is based on the following article:

- Anna M. Krol, Marvin Erdmann, Ewan Munro, Andre Luckow, and Zaid Al-Ars. "Assessing the Requirements for Industry Relevant Quantum Computation". In: *Proceedings of the 2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Montréal, Canada, 2024. arXiv: 2408.02587 [quant-ph]

CODE AVAILABILITY

A full implementation of QISS can be found at https://github.com/anneriet/QISS, see also Chapter 5.

6.1. INTRODUCTION

Industrial shift scheduling is an essential part of efficiently planning and running operations in the manufacturing sector. The challenge is to find the optimal production schedule for an end-to-end manufacturing system with multiple production sites. This schedule must comply with numerous constraints, including legal regulations and limited intermediate storage between the production sites. In volume-intensive industry sectors such as the automotive industry, one must additionally meet a production target corridor. The optimization goal is to minimize labor costs while satisfying all constraints.

The Quantum algorithm for Industrial Shift Scheduling (QISS) [115] provides the first fully quantum approach to finding exact solutions to volume-constrained industrial labor planning problems. Based on Grover Adaptive Search (GAS) [31, 66], it inherits the asymptotic quadratic speedup of Grover's algorithm over classical unstructured search methods such as brute-force or random search.

However, the problem size at which this quadratic speedup leads to a practical speedup is subject to limitations. On the one hand, it is impractical to seek exact solutions for very large problems, because: 1) the solution space grows exponentially with the problem size; and 2) the constraints typically impose very little structure on the solution space. Hence, one must resort to (classical) heuristic approaches, such as simulated annealing [107] or tensor network methods [16]. On the other hand, for sufficiently small problems where finding exact solutions may be achievable, the inferior clock speed of quantum computers compared to classical computers will tend to wash out the quadratic speedup [84]. A natural question is then: does a regime exist where QISS can return exact solutions with a runtime that is both acceptable in a real-world setting, and superior to that of classical unstructured search?

In this chapter, we investigate this question systematically by estimating the resources required for the execution of QISS (described in Section 6.2.3 below) on a fault-tolerant quantum computer using the surface code [61] for quantum error correction, as a function of problem size. Through comparison with the runtime of classical unstructured search, we evaluate the prospects for a practical speedup using QISS under various scenarios, each with different assumptions about the characteristics of the quantum computer. Using open-source resource estimation tools, we find the resources required for a speedup to be highly demanding, far out of reach for all existing and planned technology. By exploring different parameter scenarios, we quantify the role of key metrics such as qubit error rates and operation execution times in the quest for achieving a speedup.

In terms of the number of qubits, we find that currently planned quantum computers (i.e. those illustrated in Figure 6.1) will not be large enough to tackle instances of the shift scheduling problem where a classical unstructured search of the solution space becomes infeasible. This is broadly in line with the conclusions of other recent work to assess the required resources for quantum utility [84, 22, 64, 45].

The chapter is organized as follows. Background information about current quantum computing platforms, quantum error correction and the QISS algorithm is given in Section 6.2. An overview of the resource estimation process is given in Section 6.3. Section 6.7 assesses the resource requirements for the execution of QISS using quantum computers that may be accessible in the near-to-mid term, while Section 6.8 in-

vestigates the prospects for a speedup using idealized technology with fast, high-fidelity operations. Our conclusions and outlook can be found in Section 6.9.

6.2. BACKGROUND

In this section, we give background information on current and planned quantum computing platforms, quantum error correction codes, and we provide a brief overview of our use case, the QISS algorithm and the modifications we made to the QISS algorithm.

6.2.1. QUANTUM COMPUTING TECHNOLOGIES

Quantum computers are being developed by numerous academic, governmental, and industrial organizations worldwide today, covering several different types of qubit technologies. The key figures of merit characterizing these systems include: the total number of qubits, the limiting physical error rate, the degree of qubit connectivity, and the time required to execute physical operations such as quantum gates and measurements. Notably, there is currently no single platform type or device that leads across all such figures of merit.

The first two quantities (the number of qubits and the error rate) are particularly important for the goal of achieving large-scale, reliable quantum computation. Increasingly, hardware manufacturers are providing technology roadmaps for the coming 5 to 10 years of development, with the number of qubits occupying a central role. In Figure 6.1, we illustrate the number of qubits planned by different manufacturers, using publicly available data. The first devices with more than one thousand qubits are already available, while within a decade there are commitments to building devices with up to 100 000 qubits.

6.2.2. QUANTUM ERROR CORRECTION

The availability of quantum computers with hundreds of qubits, together with gate error rates around one percent, has begun to drive a wave of research in practical implementations of quantum error correction (QEC) codes [166, 58, 113, 2, 151, 24, 83]. A QEC code leverages multiple noisy physical qubits to encode a single *logical qubit*, whose error rate is lower than that of a physical qubit. For a given QEC code, the number of physical qubits required to encode a logical qubit depends on the error rate of physical qubits, as well as the required error rate of the logical qubit [70, 53, 161]. Errors on logical qubits can be detected using a set of non-destructive 'syndrome' measurements, while an accompanying decoder algorithm determines which operation(s) should be applied to correct an error. Quantum logic gates can be applied to a logical qubit by a suitable sequence of gates and measurements on the underlying physical qubits.

Quantum error correction will be essential to achieving quantum computational utility for industry-sized problems. On the other hand, QEC carries a significant resource overhead, which must be carefully quantified to predict the scale at which a computational utility may be obtained for a given use case.

In this work, our resource estimation uses the surface code as the quantum error correction scheme. The surface code is a leading candidate for large-scale fault-tolerant quantum computation, and can be readily implemented on 2D planar devices such as

superconducting qubit chips [61].

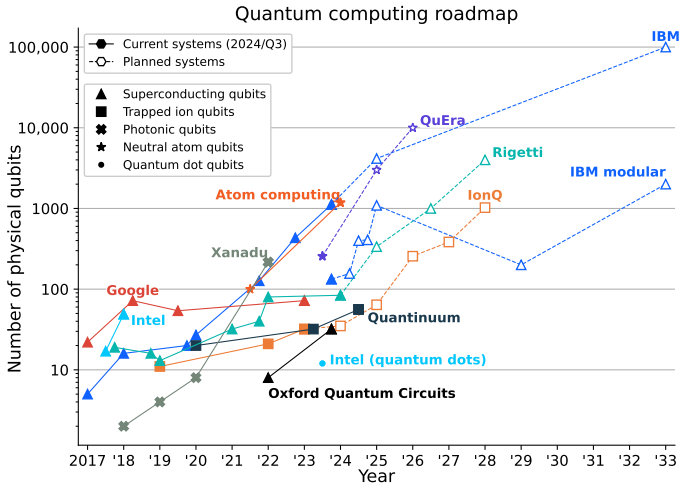


Figure 6.1: Quantum computing roadmap of the number of physical qubits by various manufacturers, inspired by [82]. Data can be found in Table 6.1.

6

6.2.3. QUANTUM OPTIMIZATION FOR INDUSTRIAL SHIFT SCHEDULING

In this section we provide a brief overview of our industrial shift scheduling problem of interest, and the algorithm designed to solve it, the Quantum Industrial Shift Scheduling algorithm (QISS); full details may be found in [115].

We consider the simplified automotive production line shown in Figure 6.2. This model consists of two *shops*: a body shop and a paint shop, with a single shared storage buffer between them. To model the effect of labor regulations, each shop has a fixed set of four possible working hours (i.e. shifts) per day. Choosing to operate a shop for a given number of hours results in a corresponding production cost and vehicle output. The intermediate storage buffer cannot be filled beyond its maximum capacity, or emptied below its minimum capacity. The goal is to find a shift schedule for each shop such that an (annual) production volume target is met, up to some tolerance, with minimal operating costs.

Figure 6.2: The structure of the simplified model for industrial shift scheduling, in which a body shop and a paint shop share a storage buffer [115]

QISS [115] uses Grover adaptive search to find the optimal solution with high probability, following a total of $O(\sqrt{N})$ applications of the corresponding Grover rotation op-

Table 6.1: Data used to generate Figure 6.1

Company	Modality	Designation	Year	# of qubits	Released?	Source	Note
IBM	Superconducting	Canary	2017	5	yes	[190]	
		Albatross	2018	16	yes	[190]	
		Penguin	2019	20	yes	[190]	
		Falcon	2020	27	yes	[190]	
		Eagle	2021 Q4	127	yes	[190]	
		Osprey	2022 Q4	433	yes	[190]	
		Condor	2023 Q4	1121	yes	[34]	
		Kookaburra	2025	4158		[190]	# of qubits from [120]
		Cockatoo	2027			[190]	# of qubits is not specified
		Blue Jay	2033	100 000		[190]	Also listed as 2000 (logical) qubits
IBM modular	Superconducting	Heron	2023 Q4	133	yes	[190]	
		Flamingo	2024	156		[190]	
		Crossbill	2024	408		[120]	
		Heron	2024	399		[190]	133 × 3
		Flamingo	2025	1092		[190]	156 × 7
		Starling	2029	200		[190]	
		Blue Jay	2033	100 000		[190]	Also listed as 2000 (logical) qubits
Google	Superconducting	Foxtail	2017	22	yes	[46]	
		Brislecone	2018 Q2	72	yes	[46]	
		Sycamore	2019 Q3	53	yes	[46]	Has 53 'effective' qubits
		M3	2025+	1000		[142]	
		M4	-	10 000		[142]	
		M5	-	100 000		[142]	
		M6	-	1 000 000		[142]	
Rigetti	Superconducting	Agave	2017 Q2	8	yes	[41]	
		Acorn	2017 Q4	19	yes	[41]	
		Aspen-1	2018 Q4	16	yes	[41]	
		Aspen-4	2019 Q1	13	yes	[41]	Higher fidelity than Aspen-1
		Aspen-7	2019 Q4	28	yes	[41]	
		Aspen-9	2021 Q1	32	yes	[41]	
		Aspen-11	2021 Q4	40	yes	[41]	
		Aspen-M-1	2022 Q1	80	yes	[41]	
		Ankaa-2	2024 Q1	84	yes	[40]	
		Lyra	2025	336		[144]	
		-	2026	1000		[144]	
		-	2028	4000		[144]	
Oxford Quantum Circuits	Superconducting	Lucy	2022	8	yes	[185]	
		OQC Toshiko	2023 Q4	32	yes	[145]	
Intel	Superconducting	-	2017 Q3	17	yes	[143]	
		Tangle lake	2018 Q1	49	yes	[86]	
	Quantum dots	Tunnel falls	2023 Q3	12	yes	[132]	
IonQ	Trapped ions	Harmony	2019	11	yes	[181]	
		Aria	2022	21	yes	[91]	
		Forte	2023	32	yes	[181]	
		Forte enterprise	2024	35		[186]	
		Tempo	2025	64		[186]	
		-	2026	256		[181]	
		-	2027	384		[181]	
		-	2028	1024		[181]	
Quantinuum	Trapped ions	H1	2020	20	yes	[155]	
		H2-1	2023 Q2	32	yes	[155]	
		H2-2	2024 Q2	56	yes	[187]	
Xanadu	Photonic	X2	2018	2	yes	[138]	
		X4	2019	4	yes	[138]	
		X8	2020 Q4	8	yes	[138]	
		X12	2020 Q4	12	yes	[150]	
		Borealis	2022 Q2	216	yes	[128]	
Atom computing	Neutral atoms	Phoenix	2021 Q3	100	yes	[38]	
		-	2024	1180		[157]	
QuEra	Neutral atoms	Aquila	2023 Q3	256	yes	[203]	
		-	2024	'>256'		[165]	Not plotted
		-	2025	'>3000'		[165]	Plotted as 3000 qubits
		-	2026	'>10 000'		[165]	Plotted as 10 000 qubits

erator. Here, N is the total size of the solution space, where for n days of factory operation we have $N = 16^n$; the constant factor of 16 represents the number of combinations of shifts for the two shops each day. In this work we use an adapted version of the QISS buffer constraint, which reduces the required number of ancilla qubits. This gives us a total qubit count of $6n + 11 + \lceil \log_2(19n) \rceil$ for scheduling n days of factory operations [115]. This is explained in more detail in Section 6.2.4.

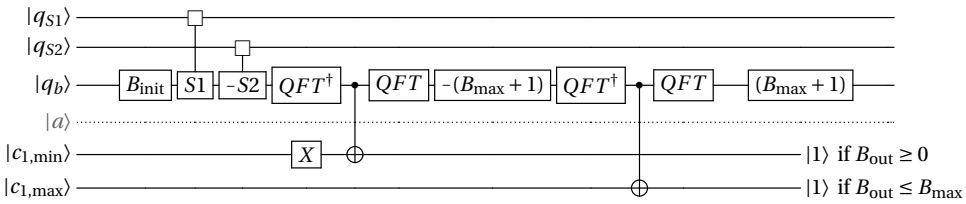
6.2.4. ADDITIONAL BUFFER CONSTRAINT

Because of (hardcoded) limitations in the quantum simulator and the resource estimation tools that we used, we need to reduce the number of qubits used by the QISS algorithm [115]. With the reduction in qubits, we can check the correctness of our algorithm for larger problem sizes than would otherwise be possible because of these limitations (see also Section 6.5).

To reduce the number of qubits required in the QISS algorithm, we introduce an additional constraint that disqualifies any solution that results in a negative number of units in the buffer. In practice, this means that shop 2, which takes units out of the buffer, is not allowed to be idle (i.e. assigned to work, but with no stock to work on).

In [115], the number of units in the buffer is set to zero if it would otherwise hold a negative number, using the $MAX(0, \tilde{B})$ operator. Because the oracle needs to be a reversible computation, the $MAX(0, \tilde{B})$ operator requires many ancilla qubits to store this negative buffer value. But if the number of units in the buffer exceeds the maximum value of B_{max}, the solution is disqualified through use of a single condition qubit per day of scheduling.

We can use the same construction for a minimum number of items in the buffer. This means that the $MAX(0, \tilde{B})$ operator and all its ancilla qubits can be replaced by a single condition qubit and corresponding constraint check. The modified circuit, shown in Circ. 6.1. This modification reduces the need for the $6n$ ancilla qubits for scheduling n days, while adding one additional condition qubit per day. This reduces the total qubit count by approximately 45%, from $11n + 11 + \lceil \log_2(19n) \rceil$ to $6n + 11 + \lceil \log_2(19n) \rceil$.



Circuit 6.1: Circuit for the evaluation of the additional minimum buffer capacity condition $B_{\text{out}} \geq 0$, with gates as defined in [115]. The CNOTs are controlled by the most significant bit (MSB) of the register $|q_b\rangle$. The buffer register is returned to its original value B_{out} through a quantum Fourier transform and by adding the value of $B_{\max} + 1$. The original formulation required the use of a register of ancilla qubits $|a\rangle$ to implement the $MAX(0, \tilde{B})$ operator. With this modification, these ancilla qubits are not required and can be omitted, at the cost of adding one additional condition qubit ($|c_{1,\min}\rangle$) per day of scheduling.

6.3. RESOURCE ESTIMATION FOR FAULT-TOLERANT QUANTUM COMPUTING

Given a target quantum algorithm and an (assumed) quantum computer, the goal of resource estimation is to quantify the required size of the quantum computer and the runtime for complete execution of the computation. The target algorithm (QISS, in this work) is provided in the form of a quantum circuit using a high-level instruction set. The high-level instructions are convenient for the construction of algorithms, but do not take into account the need for quantum error correction, or the constraints imposed by the supported instruction set and the architecture of the quantum computer.

Besides the target quantum circuit encoding the computation to be performed, one of the key inputs to resource estimation is the characteristics of the target quantum computer to be used, which is described in Section 6.3.1. At the input stage the user must also specify the error budget, which can be determined independently based on the details of the target application. The error budget strongly influences the hardware requirements, because it determines the overhead that must be introduced to perform quantum error correction. The influence and division of our error budget is explained in more detail in Section 6.3.2. Additionally, the user must make a choice for an error correction code. Section 6.3.3 describes the influence the code can have on the estimation result.

For the most accurate estimation of the required resources, the quantum circuit needs to be compiled into instructions that can be directly executed on the target hardware, in such a manner that the executable output circuit is fault-tolerant (i.e. robust to noise below a certain threshold). This compilation process consists of multiple steps, including: gate decomposition, qubit mapping and routing, resource scheduling and optimization, encoding into a quantum error correction scheme, and translation of gates into the device's native gateset [22, 45, 103]. However, at the present time many of the inputs to such a computation remain uncertain, given that quantum computers and the supporting software stack remain at an early stage of development. For quantum resource estimation, we instead make reasoned assumptions and approximations about the compilation procedure and the architecture of the quantum computer, allowing useful predictions to be obtained.

We will first explain in detail the different steps of the process in Section 6.3.4, and then give an example for a problem size of ten days in Section 6.3.5.

6.3.1. CHARACTERISTICS OF TARGET DEVICE

The key properties of the target device that can be configured in the resource estimation tools presented in Section 6.4 include the standard error rates due to gate operations, measurements, and qubit idling. Additionally, one must specify an error rate for physical T-gates, which influences the resources required for T-state distillation. When not otherwise specified, we assume T-gate error rates are the same as 1-qubit gate error rates, and that the idle error rate is the same as the measurement error rate.

6.3.2. ERROR BUDGET

The error budget is the allowed error rate after execution of the whole circuit, and represents the fraction of times the circuit is allowed to fail (i.e. to return an incorrect answer).

For Grover-based algorithms such as QISS, it is relatively easy to verify whether the solution obtained satisfies the desired search criteria. In our case, we seek a solution with an objective function value that is lower than that of any previously found solution. This means that we have a relatively high tolerance for errors in our algorithm. Therefore, we choose an error budget of $\epsilon = 0.25$ for our application.

Increasing the error budget increases the average time to solution, and decreases the chance that the optimal solution has been found after $\sqrt{N}$ application of the Grover rotation operator. We have accounted for this by setting the number of rotations to $\sqrt{N}$ rather than $\frac{\pi}{4}\sqrt{N} \approx 0.79\sqrt{N}$ for our algorithm.

The error budget is the sum of three components: the error budget for logical qubits, the budget for rotation gate synthesis, and the budget for T-state distillation. In our computations, the total error budget ϵ is divided equally among the three components, such that each has a value of $\epsilon_{\log} = \epsilon_{\text{dis}} = \epsilon_{\text{rot}} = \epsilon/3$ [22]:

1. *Logical qubits:* The error budget for logical qubits is the fraction of runs that are allowed to fail from an error in one the of logical qubits. A lower logical qubit error budget lowers the maximum allowed error rate for each logical qubit, which increases the required code distance and therefore the number of physical qubits.

2. *Rotation gate synthesis:* A lower error budget for rotation gate synthesis means an increased number of T-states required for the decomposition of each rotation gate [109], and thereby the total number of T-states required. This tends to increase the number of T-state factories and therefore the total number of physical qubits and/or the runtime of the algorithm.

3. *T-state distillation:* A lower error budget for T-state distillation means a lower maximum error for the distillation of each T-state. This influences the choice of the type of T-factory, and can mean an increase in code distance per round of distillation, an increase in the number of distillation rounds, or both.

By dividing our error budget equally, we do not take into account the runs which fail due to more than one error. Our effective error budget is thus slightly lower, at $(1 - (1 - 0.25/3)^3) \approx 23\%$. This equal division might not be the most optimal division of the error budget, since all three parts are unlikely to contribute equally to the final estimated resources. The optimal division will depend on all factors of the resource estimation, and is outside of the scope of this chapter.

We illustrate the influence of each of the three quantities in the resource estimation process in Figure 6.4.

6.3.3. QUANTUM ERROR CORRECTION CODE

A number of parameters of the chosen QEC code influence the corresponding resource requirements, including the code threshold and the crossing pre-factor. Ultimately, these determine the code distance required to satisfy the error budget for the computation. In turn, the code distance strongly influences the encoding overhead (i.e. the ratio of physical to logical qubits), and the logical cycle time (i.e. the time required to perform operations such as error detection, correction, and logical gates).

In all of our experiments we focus on the surface code, which is a leading candidate for large-scale fault-tolerant quantum computing [108, 61]. We make use of the in-built surface code models in the Azure Quantum Resource Estimator (AQRE), which are described in detail in [125].

6.3.4. PROCESS OF RESOURCE ESTIMATION

The process of quantum resource estimation can be summarized in the following five steps, shown schematically in Figure 6.3. A more elaborate schematic overview can be found in Figure 6.4.

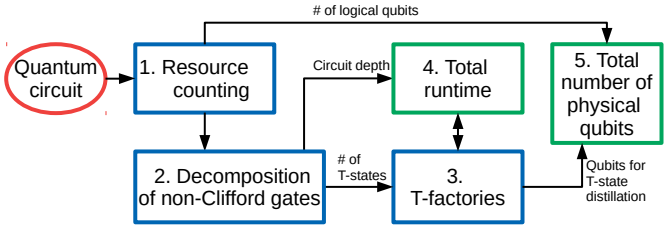


Figure 6.3: A simplified schematic overview of the process of resource estimation for fault-tolerant quantum computation. A more complete version of the diagram can be found in Figure 6.4.

1. Counting of the logical resources required by the input circuit (i.e. the circuit that encodes the target application);

2. (Abstracted) decomposition of all non-Clifford gates in the input circuit into sequences of Clifford gates and T-states;

3. Determination of the type and number of T-state factories required to supply the required T-states within the runtime of the computation;

4. Calculation of the total estimated runtime for executing the complete circuit;

5. Calculation of the total number of physical qubits required.

Note that this division of steps is a simplification of the full process of resource estimation. Other approaches may, for example, use a different subdivision of the process, or use similar steps in a different order.

1. *Logical resource counting:* The first step in the process is the circuit compilation, using the assumption that all operations in the target circuit will be implemented in practice via a standard instruction set of gates and/or measurements. For example, an arbitrary rotation around either the Y or X axis is decomposed into a product of Hadamard-type gates and a rotation around the Z axis.

Additional logical ancilla qubits are needed to facilitate interactions between logical qubits. How many ancilla qubits are needed depends on the topology of the target device, and for the most accurate estimator results, a full mapping of the input circuit onto

the target qubit layout is required. If the topology is not known or mapping is not feasible, assumptions about the qubit layout and the overhead introduced by mapping can be used instead.

For example, one may assume a 2D planar device with nearest-neighbor connectivity, with alternating rows of logical algorithm qubits (i.e. those being used explicitly for computation) and logical ancilla qubits [22].

2. *Decomposition into Clifford and T-states:* Unlike the set of Clifford gates, which in the surface code can be applied directly to logical qubits using methods of lattice surgery, non-Clifford gates must be implemented through a combination of Clifford gates and the use of *magic states* [109, 29], with the most commonly used magic state being the *T-state*.

To decompose non-Clifford gates, a sequence of Clifford gates and T-states needs to be found that can approximate the non-Clifford gate to the required fidelity. For Toffoli gates, a standard construction from the literature may be used [95]. For other non-Clifford operations such as arbitrary rotations around the Z axis, a decomposition is obtained using a recursive algorithm whose runtime grows with the specified precision [162, 48]. For large circuits with stringent precision requirements, it becomes infeasible or impractical to decompose all non-Clifford gates.

Instead, in resource estimation the error budget can be combined with existing results from the literature to calculate an average number of T-states required for the decomposition of each non-Clifford gate. The total number of required T-states can then easily be calculated by multiplying this average and the number of non-Clifford gates obtained in the resource counting step [22].

3. *Determination of the number and type of T-state factories:* To implement the quantum circuit within the specified error budget, high-fidelity T-states are prepared using *T-state factories*.

Since state injection begins with a non-error-corrected physical T-gate, the resulting logical state inherits the noise of the physical operation. Subsequently, high-quality T-states, which have a fidelity consistent with the target accuracy of the computation, are purified from multiple imperfect, noisy T-states using sophisticated quantum circuits known as *T-state distillation factories*. Different ways of implementing T-state factories have been proposed, with the type of factory chosen influencing the required number of qubits and the runtime of each factory [22, 29, 65].

4. *Estimating runtime:* The algorithm's runtime is limited by one of two parallel processes: the circuit's execution and the T-state distillation. The circuit execution time depends on the degree of parallel gate execution that is assumed possible, the specific details of the quantum error correction code used, and the execution time of underlying physical operations such as two-qubit gates, measurements, and qubit resets.

The time needed for T-state distillation also depends on these factors. However, the number of distillation factories can be chosen to either minimize the number of qubits (by using only one T-state factory) or to minimize the runtime by finding the number of factories able to produce all needed T-states at the same time as the circuit execution.

A common assumption for resource estimation is that the rate of T-state consumption is constant. In other words, we assume that the computation requires an equal number of T-states at any moment during its execution. One may relax this assumption and incorporate circuit-specific information on the T-state requirements during various stages of the computation, at the expense of introducing further modeling and computational complexity to the resource estimation procedure.

5. *Total number of physical qubits:* The total number of physical qubits required to execute the computation depends on a number of factors. Primarily, any quantum error correction scheme has an intrinsic overhead, where multiple physical qubits are used to encode a single logical qubit. The resource footprint of this encoding overhead depends strongly on the error correction code being used, the error budget of the computation, and the predominant error rate of the physical qubits.

Secondly, the algorithm design and compilation process can have a significant influence the required number of qubits. For example, the need for logical qubit routing can be reduced by designing the algorithm in such a way that each qubit only interacts with its nearest neighbors. Using more sophisticated mapping algorithms can reduce the number of physical qubits required for mapping, at the cost of increasing the runtime of the compilation process [172]. The device topology also influences the number of physical qubits. For architectures with high connectivity, the need for physical qubit routing may be reduced compared to e.g. a device with only one-dimensional (linear) qubit connectivity.

For resource estimation, we use assumptions about the influence of device connectivity and how it is used, instead of running the computationally expensive task of full mapping and scheduling of the algorithm.

The number of physical qubits for the circuit execution and the T-state distillation can be calculated separately and then added together to find the total number of qubits required.

6.3.5. ANALYTICAL EXAMPLE OF RESOURCE ESTIMATION

To get some insight into how the resource estimates are calculated, we explicitly show the calculations going into the estimations for our industrial shift scheduling algorithm for a fixed problem size of ten days, and one application of the corresponding Grover oracle. This calculation follows the steps shown in Figure 6.4. The relevant input values for the example are listed in Table 6.2, the intermediate and output values can be found in Table 6.3.

The algorithm is written as a Q# program and compiled to QIR using the standard Q# compiler built into the AQRE. We outline the steps and calculations behind the resource estimator using this example, for the “qubit_gate_ns_e3" parameter set (i.e. the ‘Flat 10^{-3}’ scenario of Table 6.6). We set the error budget to be $\epsilon = 0.001$, which we divide into three equal parts into error budgets for the logical circuit execution ϵ_{log}, the error budget for rotation gates synthesis ϵ_{syn} and the error budget for the T-state distillation ϵ_{dis}:

$$\epsilon = \epsilon_{\text{log}} + \epsilon_{\text{syn}} + \epsilon_{\text{dis}} \,, \; \epsilon_{\text{log}} = \epsilon_{\text{syn}} = \epsilon_{\text{dis}} = \frac{\epsilon}{3} = 0.000\overline{3} \tag{6.1}$$

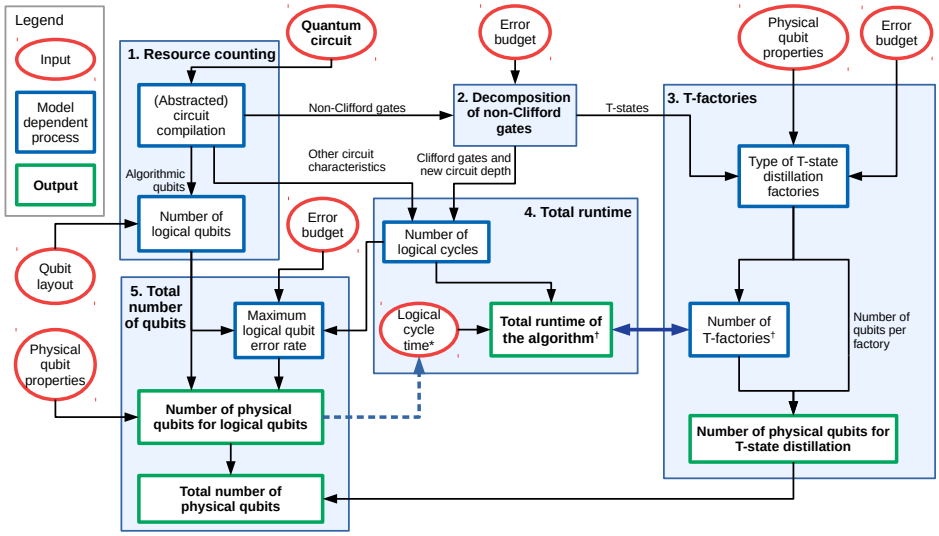


Figure 6.4: Schematic overview of the process of resource estimation as used in the Microsoft Azure Quantum Resource Estimator [22], Qualtran [154] and Bench-Q [4]. Red outlined circles show the (user) input, blue outlines mark the model dependent processes and green outlines show the output of the estimation.
* Logical cycle time can be an input value or can be calculated from the (input) gate characteristics, the code distance required to achieve the logical qubit error rate and the characteristics of the error correction code.
† The algorithm runtime can be calculated by which takes the most time: the logical circuit execution or the T-state distillation. Alternatively, the number of distillation factories can be chosen so that the T-state distillation does not increase the total runtime, but takes less than or as much time as the logical circuit execution.

The number of logical qubits and gates were obtained using the logical counts as output by the AQRE. To improve algorithm performance the AQRE uses a compilation scheme that delegates the expensive non-Clifford rotation operations to ancilla qubits. To implement a rotation gate such as $R_z(\theta)$ to a qubit, the qubit is first entangled with an ancilla by a joint Pauli measurement, after which a series of Clifford and T-states is applied to the ancilla to apply the rotation θ. By measuring the ancilla, the phase θ is kicked back to the algorithm qubit. By using this compilation technique, multiple rotations can be synthesized in parallel [22]. The ancilla qubits and measurement operations this technique requires are counted among the values listed in Table 6.2.

For a problem size of ten days, this technique adds 38 ancilla qubits to the 78 qubits needed for the algorithm execution [115], bringing the total to 116 algorithmic qubits (Q_{alg}). The additional measurement operations account for most of the counted measurements operations: of the $M_{\text{meas}} = 5858$ measurements, $4 \cdot n_{\text{days}} = 40$ are for measuring the state of the shop qubits.

The AQRE determines the number of logical qubits Q, the minimum number of logical cycles C_{min}, and the required number M of T-states as follows [22]:

Table 6.2: Input values, parameters and resource counting results for the analytical resource estimation example.

Symbol	Description	Value	Symbol	Description	Value
n_{days}	# of days to schedule	10	Q_{alg}	# of algorithmic qubits	116
ϵ	Error budget	0.001	M_{meas}	# of measurement operations	5858
p	Measurement error rate	0.001	M_{R}	# of single-qubit rotation gates	7938
p_T	T-state error rate	0.001	M_{T}	# of T-gates	912
t_{2Qgate}	Two-qubit gate time	50 ns	M_{Tof}	# of Toffoli gates	5820
t_{meas}	Measurement time	100 ns	D_{R}	Rotation depth	3653
a	Surface code parameter [22]	0.03	A	Values for determining	0.53
p^*	Surface code parameter [22]	0.01	B	the number of T-states [22, 111]	5.3

$$Q = 2Q_{\text{alg}} + \left\lceil \sqrt{8Q_{\text{alg}}} \right\rceil + 1 \tag{6.2}$$

$$C_{\text{min}} = (M_{\text{meas}} + M_{\text{R}} + M_{\text{T}}) + \left\lceil A \log_2\left(\frac{M_{\text{R}}}{\epsilon_{\text{syn}}}\right) + B \right\rceil D_{\text{R}} + 3M_{\text{Tof}} \tag{6.3}$$

$$M = \left\lceil A \log_2\left(\frac{M_{\text{R}}}{\epsilon_{\text{syn}}}\right) + B \right\rceil M_{\text{R}} + 4M_{\text{Tof}} + M_{\text{T}} \tag{6.4}$$

From Equation (6.2), we can calculate that this circuit will require $Q = 264$ logical qubits. Using Equations (6.3) and (6.4), with the values listed in Table 6.2, we calculate that the circuit will consume $M = 175\,014$ T-states and take a minimum of $C_{\text{min}} = 101\,575$ logical time steps. We will assume that the total number of logical time steps will be equal to the minimum required number for the logical circuit execution (i.e. the T-state distillation will not increase the total runtime of the algorithm). That means that we can use the number of logical cycles and the number of logical qubits Q to calculate the (maximum) logical qubit error rate P as:

$$Q \cdot C_{\text{min}} \cdot P = \epsilon_{\text{log}} \leq \epsilon/3 \tag{6.5}$$

This gives us a maximum logical qubit error rate of $P = 1.24 \cdot 10^{-11}$. Using Equation (6.6), we find that we need a (minimum) code distance of $d = 19$ to achieve the logical error rate. The required number of physical qubits per logical qubit is thus $n(d) = 722$ (Equation (6.7)), and the number of physical qubits required for encoding all of the logical qubits $q_{\text{log}} = Q \cdot n(d) = 190\,608$.

$$d = \left\lceil \frac{2\log(a/P)}{\log(p^*/p)} - 1 \right\rceil_{odd} \tag{6.6}$$

$$n(d) = 2d^2 \tag{6.7}$$

The logical cycle time for the surface code is calculated using Equation (6.8), from the two-qubit gate time t_{2Qgate} and the measurement time t_{meas}: $\tau(d) = (4 \cdot 50\,\text{ns} + 2 \cdot 100\,\text{ns}) \cdot 19 = 7600\,\text{ns}$. This gives a total algorithm runtime of $t = 7600\,\text{ns} \cdot 101\,575 = 0.772\,\text{s}$ (Equation (6.9)).

$$\tau(d) = (4 \cdot t_{2\text{Qgate}} + 2 \cdot t_{\text{meas}}) \cdot d \tag{6.8}$$

$$t = \tau(d) \cdot C_{\min} \tag{6.9}$$

Table 6.3: Intermediate and output values for the analytical resource estimation example.

Symbol	Description	Value
ϵ_{syn}	Error budget for rotation gate synthesis	$0.000\overline{3}$
$\epsilon_{\log}$	Error budget for logical circuit execution	$0.000\overline{3}$
Q	Number of logical qubits	264
$C_{\min}$	Minimum number of logical cycles	101 575
P	Logical Clifford error rate	$1.24 \cdot 10^{-11}$
d	Code distance	19
$n(d)$	Number of physical qubits per logical qubit	722
$\tau(d)$	Logical cycle time	7600 ns
ϵ_{dis}	Error budget for T-state distillation	$0.000\overline{3}$
M	Number of T-states	175 014
F	Number of T-state distillation factories	23
$\mathscr{D}_1, \mathscr{D}_{\text{last}}$	Code distance for T-state distillation rounds	5, 17
$P_T(\mathscr{D})$	Logical T-state error rate	$1.90 \cdot 10^{-9}$
$M(\mathscr{D})$	Number of T-states produced per distillation	1
$n(\mathscr{D})$	Number of qubits per factory	18000
$\tau(\mathscr{D})$	Total distillation runtime	101 μs
t	Total algorithm runtime	0.772 s
$q_{\log}$	Physical qubits for logical qubits	190608
q_{dis}	Physical qubits for T-state distillation	414000
q_{total}	Total number of physical qubits	604608

The next step is to find the number of distillation factories F capable of producing the required M T-states during the runtime of the algorithm. First, we use Equation (6.10) to calculate the required logical T-state error rate $P_T(\mathscr{D}) = 1.90 \cdot 10^{-9}$.

$$M \cdot P_T(\mathscr{D}) = \epsilon_{\text{dis}} \leq \frac{\epsilon}{3} \tag{6.10}$$

The logical output error rate of the distillation units is calculated as in Table 6.4, using the input logical T-gate error P_T and logical Clifford error P. The required output error rate of $1.90 \cdot 10^{-9}$ cannot be achieved in one round of distillation with an input T-gate error rate of 0.001, which means that multiple rounds of distillation will be needed.

We first consider the last round of distillation. To use the formula for logical output error rate from Table 6.4: $P_T(\mathscr{D}) = 35P_T^3 + 7.1P$, we initially assume that the input T-gate error (P_T) does not contribute to the output error rate of the T-factory $P_T(\mathscr{D}_{\text{last}})$, so that we can calculate the maximum input logical Clifford error rate P. We find $P \leq 1.90 \cdot 10^{-9}/7.1 = 2.68 \cdot 10^{-10}$. We use this in Equation (6.6) to find a code distance of $\mathscr{D}_{\text{last}} = 17$ for this distillation round, which we use with Equation (6.11) to calculate the actual (input) Clifford error rate $P(17) = 3 \cdot 10^{-11}$.

$$P(d) = a\left(\frac{p}{p^*}\right)^{\frac{d+1}{2}} \tag{6.11}$$

We can then use this to find the maximum allowed input T-state error rate for this round of distillation, using: $35P_T^3 \leq P_T(\mathscr{D}_{\text{last}}) - 7.1P \rightarrow P_T \leq 3.64 \cdot 10^{-4}$. This is the maximum allowed output error for the first distillation round. Many rounds of distillation can be required to reach a certain T-gate error rate, but for this example, we only need two.

Table 6.4: Characteristics of the distillation units used in the resource estimator. Table adapted from [22], with calculation for time changed to $11\tau(d)$ for consistency with resource estimator output. Characteristics are given for physical and logical space efficient and Reed-Muller (RM) preparation 15-to-1 distillation units.

distillation unit	acceptance probability	# qubits	runtime	output error rate
15-to-1 space eff. physical	$1-15p_T-356p$	12	$46t_{\text{meas}}$	$p_T(\mathscr{D}) = 35p_T^3 + 7.1p$
15-to-1 space eff. logical	$1-15P_T-356P$	$20n(d)$	$13\tau(d)$	$P_T(\mathscr{D}) = 35P_T^3 + 7.1P$
15-to-1 RM prep. physical	$1-15p_T-356p$	31	$23t_{\text{meas}}$	$p_T(\mathscr{D}) = 35p_T^3 + 7.1p$
15-to-1 RM prep. logical	$1-15P_T-356P$	$31n(d)$	$\mathbf{11\tau(d)}$	$P_T(\mathscr{D}) = 35P_T^3 + 7.1P$

For the first round of distillation, we have an input (physical) T-gate error rate $p_T = 0.001$, and an output (logical) error rate $P_T(\mathscr{D}_1) \leq 3.64 \cdot 10^{-4}$. From this, we can calculate the required logical Clifford error rate $P \leq P_T(\mathscr{D}_1)/7.1 = 5.13 \cdot 10^{-5}$ and the corresponding code distance $\mathscr{D}_1 = 5$ as before, using the output error rate in Table 6.4. The acceptance probability for this type of T-factory is $1-15P_T-356P = 0.966$. To get an output of 15 T-states with 99.9% probability, we require 18 distillation units per factory for the first round.

We can calculate the time required by both rounds of distillation using Table 6.4. For the first round, we use "space efficient" logical distillation units, and for the second round we use“RM preparation" logical distillation units. This makes the total runtime for both rounds $\tau(\mathscr{D}) = 101\,\mu$s. Each factory produces $M(\mathscr{D}) = 1$ T-state.

$$F = \left\lceil \frac{M \cdot \tau(\mathscr{D})}{M(\mathscr{D}) \cdot t} \right\rceil \tag{6.12}$$

Using Equation (6.12), this gives us $F = 23$ T-state distillation factories. The first distillation round uses $18 \cdot 20n(\mathscr{D}_1 = 5) = 18000$ qubits, while the second round of distillation uses $1 \cdot 31n(\mathscr{D}_{\text{last}} = 17) = 17918$ qubits. Each factory thus requires $n(\mathscr{D}) = \max(18000, 17918) = 18000$ qubits, and in total $q_{\text{dis}} = F \cdot n(\mathscr{D}) = 414000$ physical qubits are required for the 23 T-factories.

The total number of physical qubits is thus $q_{\text{total}} = q_{\text{log}} + q_{\text{dis}} = 190608 + 414000 = 604608$ qubits.

6.4. AUTOMATED TOOLS FOR RESOURCE ESTIMATION

We compare three different automated tools to estimate the resources required for a single day of operations in our industrial shift scheduling problem: Qualtran [154], Bench-Q [4] and the Azure Quantum Resource Estimator (AQRE, [22]). Figure 6.5 shows the

number of physical qubits and the total runtime estimated by the three tools. We analyze each case in more detail below.

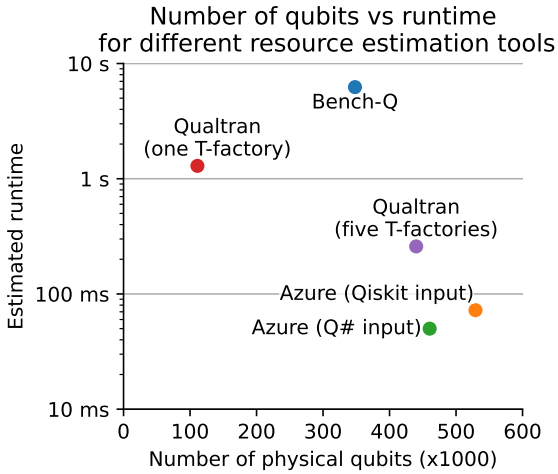


Figure 6.5: Estimated total number of physical qubits vs computational runtime for one day of operations of the industrial shift scheduling problem defined in Section 6.2.3, as estimated by Bench-Q, Qualtran and the AQRE. In all three cases, the built-in models of superconducting qubits were used, with an error rate of 0.001 and an error budget of 0.001.

6.4.1. QUALTRAN

Qualtran is a project developed by Google, consisting of a library with quantum abstractions and a rudimentary resource estimator [154]. It includes qubit, T-state factory and rotation cost models from [22] and [65]. The logical resources of our circuit were counted using the Bloq abstraction, generated from the same QASM code as used for the AQRE and Bench-Q estimates. The resulting counts were used as the input for Qualtran, using the GidneyFowler models [65], with the *SevenDigitsOfPrecisionConstantCost* rotation cost model and the *simplifiedSurfaceCode* QEC scheme. This gives the following estimation result: 111 000 physical qubits, and a total runtime of 1.3 seconds.

6.4.2. BENCH-Q

Bench-Q is a resource estimation tool developed by Zapata AI as part of the DARPA Quantum Benchmarking program [158]. It includes built-in models of trapped-ion and superconducting qubits and supports QASM input through the use of Qiskit. For the results in Figure 6.5, we used a Qiskit implementation of the industrial shift scheduling algorithm described in Section 6.2.3, with the "BASIC_SC_ARCHITECTURE_MODEL" and an error budget of 0.001. Bench-Q outputs an estimate of 348 000 physical qubits, and a total runtime of 6.2 seconds.

6.4.3. AZURE QUANTUM RESOURCE ESTIMATOR

The AQRE is a cloud-based framework developed by Microsoft that abstracts the layers of a quantum computer stack. It allows the user to customize physical characteristics such as noise parameters, gate and measurement times, the error budget, and the quantum error correction code. The input quantum circuit can be provided in either Q# or Qiskit [125]. The AQRE has been used for quantum resource estimation for various applications in [22, 45, 81].

The influence of gate decompositions and resource counting can clearly be seen in Figure 6.5 in the difference between the estimates obtained using Q# and Qiskit. For a single day of industrial shift scheduling, when using the AQRE with Qiskit as the input language we obtain an estimate of 529 000 physical qubits, and a runtime of 72 ms. On the other hand, when using Q#, the AQRE estimates 460 000 physical qubits, and a runtime of 50 ms. Using Q# produces more optimal counts for the number of logical resources, because the AQRE uses (abstractions of) optimization techniques through which some operations are implemented with additional measurements to reduce the total required number of T-states when compiling the Q# circuit. With fewer T-states, fewer factories and logical cycles are required, leading to fewer physical qubits and a shorter estimated runtime. The AQRE does not use the same optimization techniques on the circuit when it is input using Qiskit, because (integration with) this type of T-state optimization is not currently available. We expect that Qiskit can be used to produce similar results to the Q# input by tailoring the circuit optimization and gate decompositions to this specific use case.

We will use the AQRE with Q# input for our resource estimations.

6.5. EXTRAPOLATING TO $\sqrt{N}$ ITERATIONS

For full runs of Grover Adaptive Search (GAS) we need to make estimates for the resources required to run $\sqrt{N}$ iterations of Grover's search. Due to limitations in the circuit size that we were able to submit to the AQRE, we use the following method to obtain the resources required for all $\sqrt{N}$ rotations of Grover's search.

For each loop of GAS, we run a circuit with a randomly determined number of operations of the Grover rotation operator. If we want to have the same error budget for each circuit, then for longer circuits the error budget will be lower per logical gate and per logical qubit. This means more logical time steps and more physical qubits are required for these longer circuits.

The number of applications of the Grover rotation operator per loop of GAS is between zero and a number m, which is increased every time the loop does not result in an improved solution. Every time it does find an improved solution, m is reset to 1 [115]. This means that within the total of $\sqrt{N}$ iterations, we are often executing circuits with a low number of iterations, and not as many longer circuits. That the algorithm will land on a single circuit containing $\sqrt{N}$ applications of the Grover rotation operator will be rare. Doing estimates based on this longest possible circuit will give an upper bound, but in reality most runs will be for much shorter circuits. The lower bound will be given by multiplying estimates for a circuit with a single Grover rotation by the total allowed number of rotations, $\sqrt{N}$. This would be the number of resources required for a run of GAS where only circuits with one Grover rotation are selected, which might be the case

if the maximum allowed number of iterations m is reset to 1 (almost) every loop.

For getting an upper bound for the number of resources required by our algorithm, we need to run resource estimates for circuits with $\sqrt{N}$ applications of the Grover operator. Because of AQRE limits, we can only do that directly for problem sizes up to 7 days ($\sqrt{N} = 16384$)[1]. For bigger problem sizes, we instead use an approximation that we now explain.

The simplest way to make such approximations is to do the resource estimation for a single application of the Grover rotation operator for a given problem size, but reduce the error budget for this single iteration ϵ_1 such that the combined error budget for n iterations ϵ_n is equal to the error budget of a circuit that contains all $\sqrt{N}$ iterations. This scaling is shown in Equation (6.13).

$$\epsilon(n) = n \cdot \epsilon_1 = \epsilon_n \tag{6.13}$$

$$C(n,\epsilon_n) \approx n(M_{\text{meas}} + M_{\text{R}} + M_{\text{T}} + 3M_{\text{Tof}}) + \left\lceil A\log_2\left(\frac{nM_{\text{R}}}{\epsilon_1 n/3}\right) + B \right\rceil nD_{\text{R}} = nC(1,\epsilon(n)) \tag{6.14}$$

$$M(n,\epsilon_n) \approx \left\lceil A\log_2\left(\frac{n \cdot M_{\text{R}}}{n \cdot \epsilon_1/3}\right) + B \right\rceil nM_{\text{R}} + n(4M_{\text{Tof}} + M_{\text{T}}) = nM(1,\epsilon(n)) \tag{6.15}$$

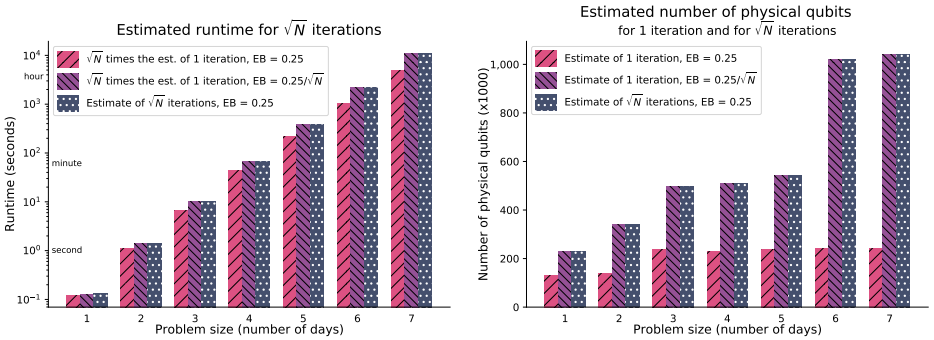
$$d(n,\epsilon_n) \approx \left\lceil \frac{2\log(3 \cdot a \cdot Q \cdot n \cdot C(1,\epsilon(n))/(n \cdot \epsilon_1)}{\log(p^*/p)} - 1 \right\rceil_{odd} = d(1,\epsilon(n)) \tag{6.16}$$

From this follow Equations (6.14) to (6.16), which are modified from the equations in [22] or Section 6.3.5. The code distance for the T-factories is essentially the same as for the logical qubits, using Equation (6.16) with $n \cdot M(1,\epsilon(n))$ instead of $n \cdot C(1,\epsilon(n)$. The rest of the estimation results follow from the results of these equations.

The two main resources we are interested in are runtime and number of qubits. For a set problem size, the estimation for the number of qubits depends only on the number of qubits required per logical qubit, and the number of qubits required for the T-factories. The first is purely dependent on the minimum code distance to achieve the required logical qubit error rate. We assume that for circuits with the same code distance, the same number and type of T-factories are selected. We further assume that the setup at the start and the measurements at the end of the circuit do not contribute much to the total circuit depth or to the error budget, so that the gate counts and depth of a circuit with n iterations are equal to n times a circuit with one application of the Grover rotation operator. And finally, we assume that the probability of errors occurring in an iteration is independent of the other iterations, and that the error budget is divided equally over all iterations.

To check the influence of these assumptions, estimations were made up to a problem size of seven days. The results of these can be found in Figure 6.6. As is clear from the plots, the estimations using the scaled error budget correspond closely to the estimates for the circuits with $\sqrt{N}$ iterations. Therefore, this is the method we will be using for all our experiments when the required circuit length exceeds the limits of the AQRE.

[1] Runs for problem sizes bigger than this number gave the following error: "It took longer than the max timeout of 10 minutes to execute one of the steps of resources estimation, so the job was stopped."



(a) Estimated runtime for $\sqrt{N}$ Grover rotations.

(b) Estimated number of physical qubits for $\sqrt{N}$ Grover rotations.

Figure 6.6: Resource estimation results for different methods of error budget (EB) scaling, for problem sizes from one to seven days. The estimates are based on resource estimator results for the superconducting qubits with a flat error rate of 10^{-3} (see Table 6.6). Estimates were made using the scenario for superconducting qubits with a flat error rate of 10^{-3} (see Table 6.6), for the QISS algorithm [115] with a single Grover rotation and a constant error budget of 0.25, for a single Grover rotation and a scaled error budget, calculated as $0.25/\sqrt{N}$, and for the algorithm containing $\sqrt{N}$ Grover rotations with a constant error budget of 0.25. To get estimates for $\sqrt{N}$ applications of the Grover rotation operator, the runtime results from a single rotation were multiplied by $\sqrt{N}$. Qubit number estimates are not multiplied, the difference between 1 and $\sqrt{N}$ iterations is caused only by the difference in error budget.

6.6. SCENARIOS FROM BEVERLAND ET AL.

Beverland et al. (2022) use six scenarios in their resource estimation: two based on superconducting qubits, two on Majorana qubits, and two based on trapped-ion qubits [22]. These scenarios are included as predefined scenarios in the AQRE. We will use these scenarios to estimate the quantum resources for our use case.

The parameters are summarized in Table 6.5, and the results are plotted in Figure 6.7. The scenarios based on trapped-ion qubits start from a problem size of 7 days. The computations for smaller problem sizes did not give results because of limitations of the resource estimator when constructing T-factories.

From the six scenarios, using the Majorana qubits with an error rate of 10^{-4} requires the highest number of physical qubits. This can be seen clearly in Figure 6.7b. More physical superconducting qubits are required than trapped-ion qubits when the error rate for both is 10^{-3}, but when the error rate is 10^{-4}, the exact same number of physical qubits are required for these two technologies. A similar number of physical qubits is required when using Majorana qubits with error rates of 10^{-6} or superconducting and trapped-ion qubits with error rates of 10^{-4}.

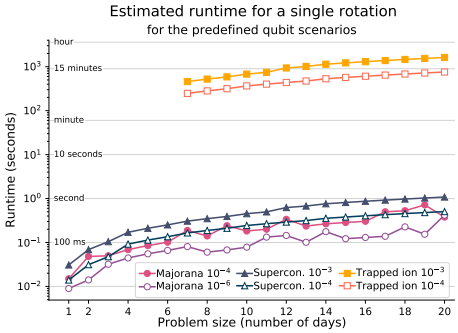
Consistent with the results in [22], trapped-ion qubits are the slowest technology by several orders of magnitude. The runtime required by the Majorana qubits is approximately the same as for the superconducting qubits. This is at odds with the results in [22], where Majorana qubits are consistently faster than other technologies. This difference could be caused by differences in problem size, error budget or characteristics of the circuit, such as the number of T-gates compared to the circuit length.

We will use scenarios based on superconducting qubits for the rest of our resource estimates for several reasons: 1) There are no current Majorana-based quantum systems that we can use to check the validity of the parameter values for the 'Majorana' scenarios; 2) the runtime of the trapped-ion qubits means it is unlikely that speedup will be achieved for this application; 3) superconducting qubits perform consistently well for this application; and 4) there are current systems with more than a thousand superconducting qubits [34, 157] and roadmaps are available for superconducting systems up to a million qubits.

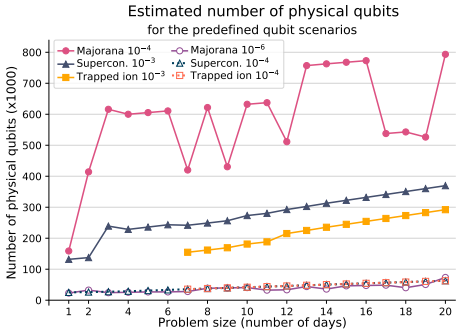
Table 6.5: Parameters for the resource estimator six qubit scenarios from [22], of which results are plotted in Figure 6.7.

Name	Major. 10^{-4}	Major. 10^{-6}	Supercond. 10^{-3}	Supercond. 10^{-4}	Trapped ion 10^{-3}	Trapped ion 10^{-4}
1Q meas. error rate	10^{-4}	10^{-6}	10^{-3}	10^{-4}	10^{-3}	10^{-4}
1Q gate error rate	-	-	10^{-3}	10^{-4}	10^{-3}	10^{-4}
2Q gate error rate*	10^{-4}	10^{-6}	10^{-3}	10^{-4}	10^{-3}	10^{-4}
T-gate error rate	10^{-2}	10^{-2}	10^{-3}	10^{-4}	10^{-6}	10^{-6}
1Q meas. time	100 ns	100 ns	100 ns	100 ns	100 μs	100 μs
1Q gate time	-	-	50 ns	50 ns	100 μs	100 μs
2Q gate time*	100 ns	100 ns	50 ns	50 ns	100 μs	100 μs
T-gate time	100 ns	100 ns	50 ns	50 ns	100 μs	100 μs

* Two-qubit joint measurements for the Majorana qubits.



(a) Estimated runtime for a single Grover rotation.



(b) Estimated number of physical qubits required for a single Grover rotation.

Figure 6.7: Resource estimator results for the different scenarios from [22], for a single Grover rotation with an error budget of 0.25. The runs for trapped-ion qubits start at a problem size of 7 days because of resource estimator limitations. The exact same number of qubits are required for the algorithm for trapped-ion or superconducting qubits with error rates of 10^4.

6.7. NEAR-TERM SUPERCONDUCTING QUBITS

We base our quantum resource estimates on superconducting qubit technology, a choice that we make for several reasons. Firstly, multiple hardware developers have long-term

roadmaps for superconducting qubit technologies, as shown in Figure 6.1. Secondly, previous quantum resource estimation results [22] have shown that superconducting qubits perform consistently well in terms of both the total computational runtime and the number of physical qubits, which we have also verified for our use-case (see Section 6.6). Finally, published characteristics of superconducting qubits are available from several different sources, allowing us to extrapolate from current technologies to make our estimates.

At the time of writing, no superconducting qubit quantum computer has surpassed the threshold for error correction with the surface code, in terms of all of the relevant physical error rates [61, 30, 199]. Therefore, the scenarios that we consider for resource estimation relate to future systems, where all figures of merit are below the threshold for QEC. Previous fault-tolerant resource estimation work has assumed all gates and operations to have identical error rates [22]. We use two of these scenarios: the superconducting qubits with 'flat' error rates of 10^{-3} and 10^{-4}. Besides these, we also consider parameter sets that better reflect the profile of existing systems.

Specifically, we construct two scenarios based on the current state-of-the-art parameters reported in [2]. The first scenario is obtained by decreasing these reference error rates by one order of magnitude; we refer to this as the 'reduced error rate' (RER) scenario. The second scenario is based on a hybrid parameter set combining the figures of merit in [2] and [35]. Concretely, [35] introduces a qubit measurement technique with a shorter duration and higher fidelity than [2], as well as an improved qubit idle error rate that we calculate from the reported decay time of 85.8 μs. The remainder of the parameters in this second scenario are drawn from [2]; in particular, to ensure that the two-qubit error rate is below the surface code threshold, we use the value without crosstalk reported in that reference. We refer to this second parameter set as the 'fast measurement' (FM) scenario. The two scenarios effectively allow us to separately probe the influence of improved error rates and faster measurements on the resources required. The parameter values for all scenarios are given in Table 6.6.

Table 6.6: Parameters corresponding to the different scenarios for near-term superconducting qubits, of which results are plotted in Figures 6.8 and 6.9. 'Error rate' is abbreviated with 'ER', values in bold are those changed compared to the reference scenario [2].

Name	Flat 10^{-3} [22]	Flat 10^{-4} [22]	Ref. [2]	RER [2]	FM [2, 35]
1Q gate ER	10^{-3}	10^{-4}	1.09×10^{-3}	$\mathbf{1.09 \times 10^{-4}}$	1.09×10^{-3}
2Q gate ER	10^{-3}	10^{-4}	6.05×10^{-3}	$\mathbf{6.05 \times 10^{-4}}$	$\mathbf{4.90 \times 10^{-3}}$
Idle ER	10^{-3}	10^{-4}	2.46×10^{-2}	$\mathbf{2.46 \times 10^{-3}}$	$\mathbf{1.63 \times 10^{-3}}$
1Q meas. ER	10^{-3}	10^{-4}	1.96×10^{-2}	$\mathbf{1.96 \times 10^{-3}}$	$\mathbf{5.00 \times 10^{-3}}$
1Q meas. time	100 ns	100 ns	500 ns	500 ns	**140 ns**
1Q gate time	50 ns	50 ns	25 ns	25 ns	25 ns
2Q gate time	50 ns	50 ns	34 ns	34 ns	34 ns

We begin by investigating the resource requirements as a function of the number of days of operation in our industrial shift scheduling problem, for a single application of the Grover rotation operator of the QISS algorithm. To interpret the results, we recall two of the basic space-time properties of the surface code. Firstly, for a surface code

of distance d, d rounds of syndrome measurements are performed to protect against measurement errors. As a result, the execution time for a logical cycle grows linearly with d. Secondly, the number of physical qubits required to construct a logical qubit grows proportional to d^2.

The results for the different scenarios are shown in Figure 6.9. All computations were performed using the Microsoft Azure Quantum Resource Estimator [22], which we have selected after comparing different resource estimation tools (see Section 6.4). The total runtime for each case has an approximately linear growth in the problem size, which can be partly explained by the linear increase of the number of logical operations in the Grover oracle [115], which is the same for all scenarios. The total runtime is determined by multiplying the number of logical cycles by the execution time for a single cycle, which in turn is a function of the gate and measurement times and the code distance. For larger problem sizes, a larger code distance is generally required to meet the error budget. However, because the code distance is always rounded up to the nearest odd number, the required code distance increases in steps and the same distance is used for a range of problem sizes, as we show in Figure 6.8. This results in the piecewise linear growth of the runtime shown in Figure 6.9.

Comparing the results for the two flat error rate scenarios, the runtime for the 10^{-4} case is roughly half that of the 10^{-3} case, because the required code distance for the former is found to be around half that of the latter (7 vs 13 and 15, see Figure 6.8). Meanwhile, the runtime for the FM scenario grows more mildly than that of the RER scenario. For the particular parameters we have chosen, the longer measurement time of RER means that its logical cycle time is always larger.

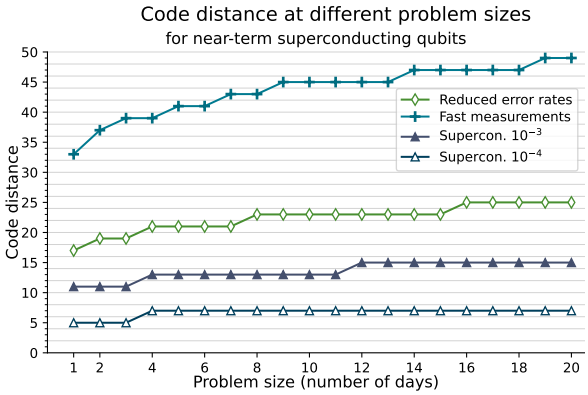
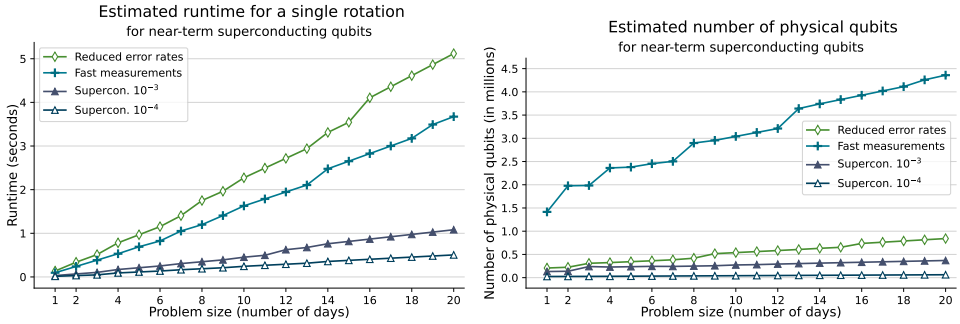


Figure 6.8: Code distance for the resource estimation of a single Grover rotation with the near-term qubit scenarios, with characteristics as outlined in Table 6.6.

The total number of physical qubits required is computed as the sum of two contributions: 1) the number of physical qubits required by the circuit itself; and 2) the number of qubits required by the T-state factories. In turn, contribution 1) has two components: one that grows linearly in the number of days, resulting from the need to encode a larger number of shift choices and to check the corresponding optimization constraints, and another that grows quadratically in the required code distance, resulting from the en-

coding of logical qubits to physical qubits. The required code distance also has a dependency on the number of days, through the error rate required to meet the error budget, as can be seen in Figure 6.8. Meanwhile, contribution 2) depends on the required code distance and on the rate at which T states are consumed. Because the number of operations in the Grover oracle increases linearly with the problem size [115], the T-state consumption rate remains constant. Overall, we find that the number of physical qubits required in each scenario also follows an approximately linear growth in the number of days, as shown in Figure 6.9.



(a) Estimated runtime for a single Grover rotation.

(b) Estimated number of physical qubits for a single Grover rotation.

Figure 6.9: Resource estimation results for the scenarios with characteristics given in Table 6.6.

From these results, it is clear that industrial shift scheduling requires a number of qubits that (far) exceeds the technology that will be available in the near future (see Figure 6.1). Furthermore, running the QISS algorithm until the best solution is found will require $\sim\sqrt{N}$ Grover's rotations [115], and for scenarios that are feasible in the near-term (such as those considered in this section), the runtime of a single Grover rotation is already on the order of seconds.

6.8. HIGH-FIDELITY QUBITS

In this section, we turn to consider the resources required to run the QISS algorithm until completion, i.e. to execute the $\sim\sqrt{N}$ Grover rotations necessary to return the optimal solution with high probability. Our goal is to identify the characteristics of a system with high-fidelity qubits that would theoretically enable a quantum speedup for the industrial shift scheduling problem. We consider five scenarios with flat error rates up to 10^{-9}, and a scenario with perfect qubits with an error rate of zero. Due to limitations of the resource estimator, we use a scaled error budget to obtain the resources required to execute the $\sqrt{N}$ iterations. This is explained in detail in Section 6.5, while the results of the resource estimations are discussed in Section 6.8.2.

6.8.1. CHARACTERISTICS OF THE HIGH-FIDELITY QUBIT SCENARIOS

In this section, we provide more detail about our choice of characteristics for our high-fidelity qubit scenarios, which are found in Table 6.7. The estimation results are plotted

in Figure 6.10.

Table 6.7: Parameters for the resource estimator for high-fidelity superconducting qubits plotted in Figures 6.10 and 6.11.

Name	10^{-3}	10^{-4}	10^{-6}	10^{-8}	10^{-9}	perfect
1Q meas. error rate	10^{-3}	10^{-4}	10^{-6}	10^{-8}	10^{-9}	0
1Q gate error rate	10^{-3}	10^{-4}	10^{-6}	10^{-8}	10^{-9}	0
2Q gate error rate	10^{-3}	10^{-4}	10^{-6}	10^{-8}	10^{-9}	0
T-gate error rate	10^{-3}	10^{-4}	10^{-6}	$\mathbf{5\times10^{-8}}$	$\mathbf{5\times10^{-8}}$	0
Idle error rate	10^{-3}	10^{-4}	10^{-6}	10^{-8}	10^{-9}	0
1Q meas. time	100 ns	100 ns	10 ns	10 ns	1 ns	0.2 ns
1Q gate time	50 ns	50 ns	5 ns	5 ns	0.5 ns	–
2Q gate time	50 ns	50 ns	5 ns	5 ns	0.5 ns	–
T-gate time	50 ns	50 ns	5 ns	5 ns	0.5 ns	–

Table 6.7 shows the error rates for the qubits used in the estimates plotted in Figure 6.11. The gate times are scaled according to the measurement time in the plot: in the first set, the measurement operation takes 1 ns and gate operations take 0.5 ns, in the second set, measurement takes 10 ns and gates take 5 ns. In the third set, measurement takes 100 ns and gate operations take 50 ns.

For the scenarios for superconducting qubits that are plotted in Figure 6.10, we use different measurement times for different error rates. The superconducting qubits used in previous resource estimations have flat error rates of 10^{-3} and 10^{-4} [22] and gate and measurement times of 50 ns and 100 ns, respectively. Based on this we set the gate/measurement time for the scenarios with error rates of 10^{-6} and 10^{-8} to 5/10 ns and for the scenario with error rate of 10^{-9} the gate/measurement time will be 0.5/1 ns.

The AQRE can only be used for qubits with error rates bigger than zero. To get estimates for perfect qubits, we used the estimated logical qubits and the logical depth as output by the AQRE. To get the runtime, the logical depth was multiplied by a value of 0.2 ns. This represents a quantum computer with a clock frequency of 5 GHz, that does not use any error correcting methods but does take some mapping and gate decompositions into account.

When the gate error rate is below a certain limit compared to the required logical T-gate error rate, the AQRE cannot calculate the number and type of T-factories. To circumvent this, we set the minimum T-gate error to $5 \cdot 10^{-8}$. Even with this, no estimates could be made for problem sizes below 4 days for the scenarios with error rates of 10^{-8} and 10^{-9}, and below 2 days for the scenario with error rate of 10^{-6}. These are shown as missing data points in Figure 6.10.

For the brute-force search, we set the time required for a checking a single guess at 1 ns. This is much shorter than it takes for an actual guess for the best solution, which includes calculations and requirement checks. The value of 1 ns was chosen to represent a parallel implementation of the search, where a new guess is completed (on average) every nanosecond. How long an actual brute-force search takes depends on many factors, including quality of implementation and resources available for parallel execution. The runtime plotted here serves as an indication for the exponentially increasing classical execution time.

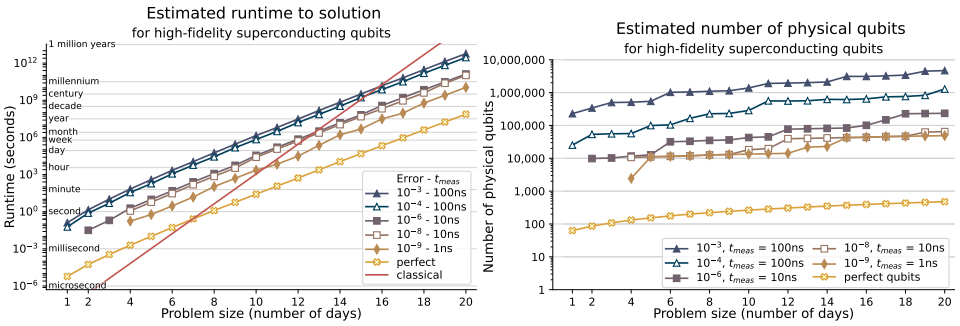
When decreasing gate error rates to 10^{-8} or 10^{-9}, we reach the limits of what the AQRE can be used for. At these error rates, the AQRE cannot calculate T-factories. Decreasing the error rate for T-gates further is not possible, and the estimation for the number of physical qubits required is most likely fully determined by the T-gate error and factors that we have no influence over. It is likely that a real quantum computer with such low qubit error rates requires less qubits than are estimated here and has other limiting factors that we cannot predict from current technologies.

6.8.2. RESULTS FOR THE HIGH-FIDELITY QUBIT SCENARIOS

The flat error rate scenarios investigated in the previous section had gate error rates of 10^{-3} and 10^{-4}, gate execution times of 50 ns, and measurement times of 100 ns [22]. We consider three additional high-fidelity scenarios, with error rates of 10^{-6}, 10^{-8} and 10^{-9}. We further assume that advances in qubit technology will lead to an improvement in all figures of merit simultaneously, taking the measurement time to be 10 ns for the 10^{-6} and 10^{-8} scenarios and 1 ns for the 10^{-9} scenario. The execution time for all gate types is half of the specified measurement time, i.e. 5 ns and 0.5 ns. Our assumptions, the construction and execution of the computations for these high-fidelity scenarios are discussed in more detail in Section 6.8.1.

The results of the resource estimation with the high-fidelity scenarios are shown in Figure 6.10. We compare the runtime with a classical brute-force search, assuming a set of parallel processes with a combined completion frequency of 1 GHz, meaning that 10^9 guesses at the best solution are completed every second. For error rates of 10^{-3} and 10^{-4}, the number of physical qubits is of the order of a few million for problem sizes up to 20 days. However, the quantum algorithm does not exhibit a speedup over classical brute-force search until the complete computation takes a over a century. For error rates between 10^{-6} and 10^{-8} with a measurement time of 10 ns, the quantum algorithm becomes faster than the classical brute-force search around a problem size of 12 days, with execution times on the order of weeks. For a very low physical error rate of 10^{-9}, the quantum algorithm achieves a speedup for scheduling 11 days, taking a little over 2 hours compared to almost 5 hours using the classical approach. A quantum computer with completely error-free qubits would enable a speedup over classical brute-force search for problem sizes larger than 7 days. However, even in this case the total runtime would be of the order of months for scheduling 20 days of operations.

To further investigate the influence of error rates and measurement execution times, we focus on the particular case of scheduling for 12 days of operation. We consider different combinations of the two system figures of merit, and plot the runtime and the number of physical qubits in Figure 6.11. When a measurement takes only 1 ns, all of the tested scenarios are faster than the brute-force approach for this problem size, except the one with an error rate of 10^{-3}. When measurements take 10 ns or more, all scenarios are slower than the classical approach. These results clearly show that the error rate has less influence on the overall runtime than the measurement execution time. For example, decreasing the error rate from 10^{-3} to 10^{-9} gives a speedup of approximately an order of magnitude, whereas a reduction in measurement time translates directly into the same reduction in total runtime.



(a) Estimated runtime for arriving at the best solution.

(b) Estimated number of qubits for arriving at the best solution.

Figure 6.10: Resource estimation results for different scenarios based on high-fidelity superconducting qubits, with error rates from 10^{-3} to 10^{-9} and measurement times of 100 ns to 1 ns. The execution time for all gate types is half of the specified measurement time. Full characteristics can be found in Table 6.7.

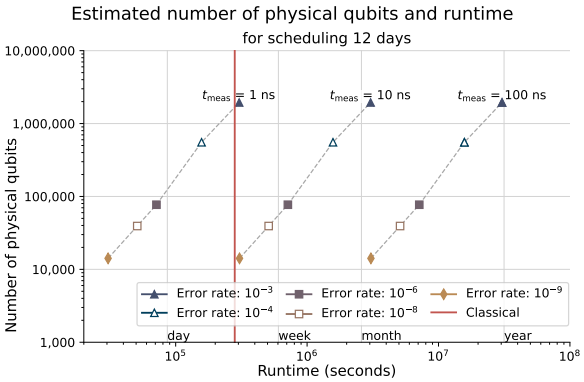


Figure 6.11: Resource estimator results for scheduling twelve days for superconducting qubits error rates from 10^{-3} to 10^{-9} and measurement times of 1 ns, 10 ns and 100 ns. The execution time for all gate types is half of the specified measurement time. Full characteristics can be found in Table 6.7.

6.9. CONCLUSION

This work was motivated by the question of whether a speedup could realistically be achieved for industrial shift scheduling using the QISS algorithm. We have compared a number of scenarios with different system parameters, covering quantum computers that could feasibly be available in the near or medium term, as well as hypothetical high-fidelity scenarios that would require significant technological improvements. We find conclusively that the near-term scenarios will not deliver a quantum speedup, while even the high-fidelity scenarios would require execution time of measurement operations to be below 10 ns. This is in line with existing work investigating the plausibility of quantum speedups using Grover-based algorithms, which emphasized that the slow execution time of quantum operations compared to classical ones is a major obstacle for such approaches [84, 14].

Besides hardware improvements to reduce gate and measurement times, there are two further areas where future work could significantly change the prospects for a quantum speedup for large industrial optimization problems. Firstly, novel quantum error correction schemes could result in a faster logical clock cycle, which could occur if, for example, the required code distance is smaller. In turn, this could be achieved for QEC codes with a higher threshold than the surface code, although we emphasize that the cycle time for a QEC code is determined by many other factors besides its threshold. Secondly, advances in quantum algorithms could lead to approaches that achieve beyond-quadratic speedups, potentially significantly reducing the crossover point at which a practical quantum speedup can be realized.

For future work, one can consider interesting experimentation by executing small instances of the QISS algorithm on quantum computers that will be available within the next decade. For example, the results of Figure 6.10 suggest that in the 10^{-3} and 10^{-4} flat error rate scenarios (which may be considered realistic with hardware improvements) one could execute the algorithm on problem sizes of around 7 to 9 days, with a runtime in the order of hours or days. It may then be of interest to systematically study the impact of executing fewer than the maximum number of Grover iterations, in order to determine the trade-off in runtime and solution quality. With fewer iterations the probability of obtaining the optimal solution is reduced, however in practice one may be interested instead in obtaining a good approximate solution. In this context, one could investigate the probability of obtaining a solution satisfying some acceptable approximation ratio, as a function of the number of iterations and the number of trials, where independent trials may also be parallelized over multiple quantum processors. It may then be of interest to compare the performance of this heuristic quantum approach to that of classical heuristics.

7

CONCLUSION

Because of recent stagnating single-thread performance and limited potential for further miniaturization of transistors, the computing industry is looking towards new technologies as the basis for the next generation of computing. One of these new technologies is quantum computing. Current quantum hardware does not yet fulfill the criteria for building a quantum computer that achieves quantum supremacy, quantum utility or quantum advantage.

Besides improvements in quantum hardware, the complete quantum computing stack will also need to be made ready for programming of millions of qubits, which is not feasible with current quantum programming languages. This gives us our main research question:

- How can we make the quantum computing stack ready for utility-scale quantum computing?

We have split this into the following five research questions:

1. What are useful high-level abstractions for programming quantum computers?
2. How can existing high-level abstractions such as unitary decomposition be improved?
3. Can the performance of quantum algorithms be improved through compiler optimizations?
4. What does a quantum algorithm for a real-world use-case look like?
5. What are the requirements for practical quantum computing?

We have developed answers to these questions over the course of this dissertation. For each question, we will summarize our findings and provide an outlook for future research directions.

1. What are useful high-level abstractions for programming quantum computers?

This dissertation introduces a number of effective high-level abstractions to implement quantum computing applications. These abstractions have also been use in the implementation of the algorithms presented in this dissertation, represented as high-level quantum operators and functions. In the following, we will list these high-level abstractions, split between the different application domains covered in this dissertation.

In the circuit construction for unitary decomposition algorithms, many of the same set of (multi-qubit) operators are often used. These are:

- Generic single and multi-qubit gates, which can be decomposed using a unitary decomposition method.
- Multi-qubit gates with a single control qubit. This encompasses both controlled arbitrary (multi-qubit) gates, where the gate is only applied if the control qubit is in state $|1\rangle$, and quantum multiplexors or uniformly controlled quantum gates, where different gates are applied based on the state of the controlling qubit. Both can be decomposed using quantum demultiplexing.
- Uniformly controlled (rotation) gates, which can be decomposed with the Gray code method if the gate is a rotation gate.
- State preparation, which can be implemented using a decomposition method for generic quantum gates or a method tailored to state preparation.

For hybrid classical-quantum algorithms, such as variational quantum eigensolvers, we have implemented the following abstraction:

- (Explicit) parameterization of the circuit, which can be used with classical optimizers to update the relevant values for each iteration of the algorithm.

For Grover's adaptive search, Grover's algorithm and constraint-bound (integer) optimization problems like the quantum industrial shift scheduling algorithm, the following set of abstractions was used:

- Grover's adaptive search, which is a hybrid algorithm which required support for using the (parameterised) quantum algorithm within a classical loop.
- Constructs used in Grover's algorithm, such as equal superposition of all the states in the solution space, applying the inverse of a user-defined operator (such as the oracle), the diffusion operator and measurement of multiple qubits.
- Storing encoded (binary) numbers using qubit registers.
- Quantum adders in the Fourier basis, which uses phase-gates controlled by multiple qubits.
- Quantum Fourier transforms and inverse quantum Fourier transforms.

- Setting a qubit register to $|0\rangle$ when the encoded number is below zero.
- Flipping condition qubits to $|1\rangle$ when a number encoded on a qubit register is below or above a certain value, which might be updated between iterations of the algorithm.
- Using a specific output qubit in $|-\rangle$ which is used to apply the phase shift to specific marked qubit states.

Future research directions:

- Some of the high-level abstractions listed above are already available in the quantum programming languages we used for our implementations (OpenQL, Q# and Qiskit), but many are not. Each abstraction that was not already available needed to be implemented and debugged before we could start implementing the overall algorithm.

  The list of abstractions above can thus be used to develop a (domain-specific) quantum programming language or function library. With such a programming language or library, developing new quantum algorithms will be quicker and easier, which will enable the development of more sophisticated, larger-scale algorithms than is feasible to program with any currently available quantum programming language.

2. How can existing high-level abstractions such as unitary decomposition be improved?

With our block-ZXZ based decomposition method, we show how we can decompose an arbitrary n-qubit gate into at most $\frac{22}{48}4^n - \frac{3}{2}2^n + \frac{5}{3}$ CNOT gates. This is $(4^{n-2}-1)/3$ less than the best previously published work [173]. More specifically, we can construct a general three-qubit operator with at most 19 qubits, which is currently the least known for any exact decomposition method.

To arrive at this decomposition algorithm, we used the optimizations presented by [139] and [173], gate commutation and gate merging to optimize the block-ZXZ decomposition [50].

When implementing unitary decomposition in OpenQL, two optimizations were implemented to take advantage of the internal structure of the input or intermediate unitary matrices, which can drastically reduce the length of the resulting circuit. With these optimizations, the final gate count can be much lower than the numbers predicted by the formula, which are the maximum possible gate count for a given decomposition method.

Future research directions:

- Our block-ZXZ based decomposition follows the same structure as the quantum Shannon decomposition, and has the same benefit of using recursion on generic quantum gates. This means that the decomposition can take advantage of the known optimal decompositions for two-qubit unitary gates, and other small-circuit optimizations, heuristic methods or optimal decompositions for three or more qubit gates when these become available.

- Other than general unitary gates, the decomposition uses only single qubit gates and diagonal gates. This simplifies the structure and presents further opportunity for optimization down the line, such as accounting for specific hardware constraints like connectivity when decomposing the diagonal gates.

- The circuit output of the decomposition can be compiled to any universal gateset. The resulting circuit will have the same overall structure with an equal number of two-qubit gates when the gateset includes a two-qubit gate that is equivalent to the CNOT gate up to single qubit gates.

- If these circuits are executed on a quantum execution platform which has a more permissive gateset, the diagonal gates can also be implemented with uniformly controlled Z-gates [141] instead of CNOTs.

- Other decomposition methods, like the QR, QSD and Cartan decompositions, have been generalized to higher-dimensional quantum systems [94], which may offer practical advantage over two-level qubits [119]. If our decomposition is also generalizable to multi-level quantum systems, it may result in more optimal gate counts for these types of systems as well.

- For simple controlled arbitrary multi-qubit gates, there might be a more optimal decomposition method than quantum demultiplexing, since these types of gates have much fewer degrees of freedom than quantum multiplexors. If a more efficient method is found, it would reduce the total gate count resulting from the overall decomposition.

3. Can the performance of quantum algorithms be improved through compiler optimizations?

We have implemented a unitary decomposition method and efficient parameterization in the OpenQL programming framework. With the implementation of unitary decomposition, OpenQL can now be used for any quantum algorithm that uses arbitrary unitary gates. One such algorithm is QiBAM [167], which cannot be implemented without unitary decomposition.

We have compared our implementation of unitary decomposition to Qubiter and to the UniversalQCompiler. Our implementation is based on (unoptimized) quantum Shannon decomposition, which is up to 10 times more efficient in the number of generated gates than Qubiter and only 50% less efficient than the implementation of UQC. Although the execution time of the decomposition is $O(8^n)$ for matrices of size $2^n \times 2^n$, for the decomposition of up to 10-qubit gates, our implementation is 10 to 100 times faster than Qubiter and about 500 times faster than the implementation in UQC.

We have also introduced OpenQL$_{\text{PC}}$, an efficient approach to parametric compilation for hybrid quantum-classical algorithms, and implemented it in OpenQL. OpenQL$_{\text{PC}}$ is designed to be modular, scalable, usable, future-proof and fast. We compared wall-clock compilation time of OpenQL$_{\text{PC}}$ with OpenQL, Qiskit and PyQuil. The total compilation time was measured using the MAXCUT benchmark from [133].

Experimental results show that for the compilation of MAXCUT circuits, the total compile times of OpenQL and OpenQL$_{\text{PC}}$ are between 10 and 20 times faster than Qiskit,

respectively. PyQuil has the slowest compilation, about 60 times longer than OpenQL$_{PC}$. In addition, comparing compile times of multiple compile iterations relative to single iterations shows that OpenQL$_{PC}$ is the fastest, followed by Qiskit which is 1.2x slower, while OpenQL took the most time being 1.8x slower than OpenQL$_{PC}$.

Tests were also done for the complete MAXCUT benchmark, which includes a classical optimizer and simulation of the circuit in each iteration. These tests, with shorter circuits but more iterations than before, show that the improvements made in OpenQL$_{PC}$ result in a decrease of 40 to 70% in (accumulated) compilation time. This makes OpenQL$_{PC}$ faster than all other tested languages. For the total execution time of the MAXCUT benchmark, the performance of the simulators has more influence than the compile times of the languages. The simulator used with PyQuil is still the slowest option by far, but the Qiskit Aer simulator is 2 to 3 times as fast as the QX simulator used by OpenQL and OpenQL$_{PC}$. Even so, the faster compile time of OpenQL$_{PC}$ does lead to a clear speed-up of the complete benchmark.

This clearly shows that compiler optimizations can be used to improve the performance of quantum algorithms. Furthermore, a single compilation of a quantum circuit is projected to become more computationally expensive as more sophisticated mapping, optimization and error-correcting algorithms are created and implemented, which will further increase the cost of repeated compilations in hybrid algorithms and increase the impact of our approach.

Future research directions:

- The implementation of unitary decomposition in OpenQL can be updated to use the block-ZXZ based decomposition method presented in Chapter 2, which can generate a circuit with fewer CNOT gates than is possible with QSD. Both decompositions have the same high-level structure, which means that the circuit-level and execution time optimizations presented in this dissertation will still provide the same, or similar, benefit with the new decomposition method.

- When the presented method of efficient parameterization is extended to and integrated in quantum systems and simulators, it can be used to decrease the data transfer for hybrid algorithms, which will improve the latency bottleneck caused by the movement of data over the entire stack of HPC and QPU systems. Because we define the static and dynamic parts of an algorithm explicitly, only the updated parameter values need to be transferred to the classical control systems of the QPU. To implement such an extension requires explicit access to these classical control systems.

4. What does a quantum algorithm for a real-world use-case look like?
In a collaboration with BMW, Germany and Entropica Labs, Singapore, we have developed the Quantum Industrial Shift Scheduling algorithm (QISS), a quantum algorithm based on Grover's adaptive search for industrial shift scheduling problems with production target and intermediary storage constraints, a situation found in settings such as the automotive industry.

We show the construction of a quantum circuit to implement the necessary Grover's oracle for an arbitrary number of days of factory operations, within the context of a sim-

plified model comprising two shops and one buffer. For small problem instances, we have numerically corroborated the performance of the algorithm. In particular, in our examples we verify that $\sqrt{N}$ applications of Grover's rotation operator suffice to find the optimal solution, where N is the size of the solution space. This shows that an asymptotic quadratic speedup can in principle be achieved over classical unstructured search.

In practice, this may be useful in at least two scenarios: (1) to obtain exact solutions to problems of small or modest size, in the context of benchmarking heuristic algorithms designed to scale to much larger, industrial problem sizes; (2) to use QISS as an integral component of a heuristic strategy, where exact solutions for short time periods are used to construct a solution for a longer time period.

Future research directions:
Our work lays the foundation for future research in several directions.

- In the context of the simplified model, QISS could be used as a target algorithm to study and benchmark the performance of quantum computing systems. Specifically, in the pre-fault-tolerant era one could test the different circuit primitives, and investigate their susceptibility to noise on different quantum computing platforms. Techniques to suppress, mitigate, or detect certain types of error may be found, which could allow for improved results, and yield useful insights for running the algorithm in the future on fully fault-tolerant machines. In the fault-tolerant era, QISS could be incorporated into application-level benchmark frameworks [60].

- We have focused on a simplified shift scheduling model with only two shops and a single buffer. It would be interesting to develop Grover's oracles for more complex, more realistic instances of industrial shift scheduling. For example, new circuit primitives may need to be designed due to the intricacies resulting from multiple and/or shared buffers (see Section 5.3.4). Moreover, while the resulting circuits would very quickly become intractable to simulate on a classical computer, even for a single day, it would allow a quantitative investigation of the additional circuit complexity arising from the presence of multiple shops and buffers.

- In Chapter 5 we have developed quantum circuits at the *logical* level, which assumes qubits to be perfectly noiseless. To implement these circuits in practice on quantum computers, which are always subject to some degree of noise, a fault-tolerant quantum error correction (QEC) scheme would be necessary. QEC introduces significant resource overheads in terms of the number of required qubits and gate operations. Key quantities of interest such as the total number of *physical* qubits and the total computational runtime can be estimated through frameworks incorporating different layers of assumptions on the architecture of a fault-tolerant quantum computer, and the compilation of quantum programs [22, 154]. Subsequently, one could compare these estimates to those of classical computing approaches, and seek to identify the scale at which QISS may deliver a speedup in practice for unstructured search, taking into account the gap in execution time for basic quantum and classical circuit operations [14, 84].

- Finally, we have not investigated whether the industrial shift scheduling problem is amenable to solution by dynamic programming methods, however we note that quantum algorithms capable of exploiting optimal substructure have been described in [9]. Therefore, if a speedup over unstructured search can be obtained classically with dynamic programming, it may also be possible to incorporate this into QISS.

5. What are the requirements for practical quantum computing?
We have approached this question through the QISS algorithm, and by evaluating whether a speedup could realistically be achieved for industrial shift scheduling using QISS. We have compared a number of scenarios with different system parameters, covering quantum computers that could feasibly be available in the near or medium term, as well as hypothetical high-fidelity scenarios that would require significant technological improvements.

We find conclusively that the near-term scenarios will not deliver a quantum speedup, while even the high-fidelity scenarios would require that measurement operations take at most 10 ns. This is in line with existing publications investigating the plausibility of quantum speedups using Grover-based algorithms, which emphasize that the long execution time of quantum operations compared to classical ones is a major obstacle for such approaches [84, 14]. As we have seen in Figure 1.3, for a quantum algorithm with cubic, quartic or super-polynomial speedup (such as Shor's), the execution time of quantum operations is less critical but will still need to be improved before a quantum computer will be faster than its classical counterpart.

Future research directions:
Besides hardware improvements to reduce gate and measurement times, there are two further areas where future work could significantly change the prospects for a quantum speedup for large industrial optimization problems.

- Firstly, novel QEC schemes could result in a faster logical clock cycle, which could occur if, for example, the required code distance is smaller. This could be achieved by using a (novel) QEC scheme with a higher error threshold than the surface code, although we emphasize that the cycle time for a QEC code is determined by many other factors besides its threshold.

- Secondly, advances in quantum algorithms could lead to approaches that achieve beyond-quadratic speedups, potentially significantly reducing the crossover point at which a practical quantum speedup can be realized.

For future work that builds more directly upon the work presented in Chapter 6, one can consider interesting experimentation by executing small instances of the QISS algorithm on quantum computers that will be available within the next decade. For example, the results of Figure 6.10 suggest that in the 10^{-3} and 10^{-4} flat error rate scenarios (which may be considered realistic with hardware improvements) one could execute the algorithm on problem sizes of around 7 to 9 days, with a runtime in the order of hours or days. It may then be of interest to systematically study the impact of executing fewer than the maximum number of Grover iterations, in order to determine

the trade-off in runtime and solution quality. With fewer iterations the probability of obtaining the optimal solution is reduced, however in practice one may be interested instead in obtaining a good approximate solution. In this context, one could investigate the probability of obtaining a solution satisfying some acceptable approximation ratio, as a function of the number of iterations and the number of trials, where independent trials may also be parallelized over multiple quantum processors. It may then be of interest to compare the performance of this heuristic quantum approach to that of classical heuristics.

This brings us back to the main research question: **How can we make the quantum computing stack ready for utility-scale quantum computing?**
For the quantum stack to be ready for utility-scale quantum computing, several major improvements will need to be made to prepare for programming and compiling circuits with millions of qubits.

- We will need high-level abstractions that will speed up programming of quantum computers, allow for (easier) debugging and will allow for programming millions of qubits.
- The classical component of the compilation and compute of (hybrid) quantum algorithms will need to be improved.
- More algorithms for real-world use-cases will need to be developed, which will provide a basis for improvements across the quantum stack that will lead to quantum utility.
- We need to do quantum resource estimation for real use-cases, in order to have insights into what utility-scale quantum computing will look like.

ACKNOWLEDGMENTS

Thank you to everyone who helped me on this journey. I could never have done this without you. There are many more people than those I could thank here, so I would also like to extend my thanks to everyone who helped me along the way even if your name is not on this list.

First and foremost, I would like to thank Zaid Al-Ars, my supervisor, mentor and promotor, who took me under his wing when I was suddenly without a PhD supervisor. Thank you for all the weekly meetings, for always giving me the confidence, motivation and guidance on how to continue my research. Thank you for your unwavering enthusiasm in my ideas and in my work and for not giving up on me when I hit some dark times during the pandemic winters. And thank you for your willingness to correct some truly terrible first drafts and for teaching me the importance of catchy titles and nice graphics.

I also need to thank Peter Hofstee, who was often geographically far away but closely involved when it counted. Thank you for your valuable insight during our meetings, and especially for encouraging me to do an internship with BMW in Munich. It was a great addition to my PhD and a fantastic experience that I will carry with me in any future endeavors.

Thanks to Koen Bertels, who supervised my Master's thesis and who started me on this PhD journey. When I came to you to ask about doing my Master's thesis with you, you immediately started encouraging me to do a PhD as well. Thank you for your unwavering optimism and enthusiasm during my Master's thesis, which helped to convince me to give this whole PhD thing a try.

Thanks to Aritra Sarkar, who started me on unitary decomposition during my Master's thesis and who was my fellow PhD under Zaid. Thank you for your insights in our weekly meetings and beyond, and for always being available to answer my PhD related questions, both the technical ones and my many questions about the TUDelft bureaucracy.

Thanks to everyone I worked with and met during my internship at BMW in Munich. Thanks to Marvin Erdmann, my supervisor at BMW, for his encouragement, his advice and his insights. We still need to play that game of DnD sometime. Thanks to Ewan Munro, Madrid-based CTO of Entropica labs, Singapore, for his collaboration on and contributions to the two papers we wrote together, which became of much higher quality through his feedback, his in-depth questions, his scientific curiosity and his (re)writing of certain sections. Although we ended up in the muck of discussions sometimes, I have learned a lot through (re)examining, reasoning through and occasionally defending my work (and thanks to Marvin for intervening and mediating when necessary). Thanks to Andre Luckow, for asking great questions and heading the Emerging Technologies department, which includes many great people that helped make my BMW internship a really great experience for me: Jinyu Lee, Youran Song, Hyerim Park, Rudi Finžgar, Leo

Hölscher, Florian Kiwit, Ann Christin Rathje, Lukas Müller, Philipp Ross, Chris Wittig, Marwa Marso, Carlos Riofrio, Max Passek, Chris Wittig and others.

And thanks to everyone I worked with during my time at the TUDelft, the interns (Neil Eelman), MSc students (Koen Mesman, Huub Donkers, Smaran Adarsh). Thanks to Matthias Möller for the meetings and discussions. Thanks to Sebastian Feld and his group, who I also had a lovely time in Seattle with: Medina Bandic, Nikiforos Paraskevopoulos, Matthew Steinberg, Luise Prielinger and Pablo le Henaff, who I also want to thank for his hard work on OpenQL. My fellow TAs and office mates: Christiaan Boerkamp and Yongding Tian. And the support staff of the QCE department, which includes Francis Bulters, Paul de Wit, Trisha de Jonge, Laura de Groot, among others. And thanks to all other TUDelft student and employees that I interacted with over the years, even something as "simple" as explaining my PhD topic has provided me with new motivation, ideas and clarity over the years.

Thanks to Pepijn, for being such a good friend, through all the good times and the bad, for being better than me at Mario Kart and cooking (and many other things). You are an amazing person and I am so, so, so glad that we are friends. Thanks to the "DnD guys": Walewein, Alies, Anne, Jelte, Roemer and Wietske, to my old flatmate and friend Josanne, to my oldest friends: Lotte and Fien, to the "space girls": Livia, Marleen and Sascha, to my gilde Momus and my yearclub Zephyros. And thank you to all my other friends as well, who I will not list because I cannot name you all, but please consider yourself thanked.

Thanks to my old cat Beau, may you spend your time hunting paper scraps and laying in the sun in cat heaven, and my ~~Tinerfeños~~ ~~Müchner~~ Delftse cats: Charlie and Lana, you may not have a braincell but you are excellent lap-warmers. Thank you to Sophie von Boeckmann, from the Tierhilfe, and Annina Angene, their foster mama, for taking care of them before me. And additional thanks to my friends who I could count on to care for my cats while I was away on holiday or at a conference: Pepijn, Josanne, Alies, Walewein, Youran and Jinyu.

Thanks to my siblings and their families: Gerard, Nynke, Jits, Fas, Bas and little Lucy, Mayke and the baby. Thank you for your support, your help and all the great discussions over the breakfast, lunch and dinner table. I may have bragged a few times over the years about how amazing and cool you all are, and I consider myself extremely lucky to be related to you all.

And thanks to my parents, Adri and Jacqueline, for always encouraging me, listening to me talk about all my latest interest, for coming to all my violin recitals, for driving me to and from Munich and for everything else. I never could have done this without you.

BIBLIOGRAPHY

[1] Benoît Jacob (founder), Gaël Guennebaud (guru), and many more. *The Eigen documentation*. Accessed on: 20-07-2020. 2019. URL: http://eigen.tuxfamily.org/index.php.

[2] Rajeev Acharya et al. *Suppressing quantum errors by scaling a surface code logical qubit*. 2023. DOI: 10.1038/s41586-022-05434-1.

[3] Alfred V. Aho and Krysta M. Svore. *Compiling Quantum Circuits using the Palindrome Transform*. 2003. arXiv: quant-ph/0311008 [quant-ph].

[4] Zapata AI. *Bench-Q*. 2024. URL: https://github.com/zapatacomputing/benchq (visited on 04/2024).

[5] G. Donald Allen. "Unitary Matrices". In: *Lectures on Linear Algebra and Matrices*. College Station, TX: Texas A&M University, 2003. Chap. 4, pp. 157–180.

[6] C. G. Almudever, L. Lao, R. Wille, and G. G. Guerreschi. "Realizing Quantum Algorithms on Real Quantum Computing Devices". In: *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2020, pp. 864–872.

[7] Esteban Álvarez, Juan-Carlos Ferrer, Juan Carlos Muñoz, and César Augusto Henao. "Efficient shift scheduling with multiple breaks for full-time employees: A retail industry case". In: *Computers & Industrial Engineering* 150 (2020), p. 106884. ISSN: 0360-8352. DOI: 10.1016/j.cie.2020.106884.

[8] Andris Ambainis. *Quantum search algorithms*. 2005. arXiv: quant-ph/0504012 [quant-ph].

[9] Andris Ambainis, Kaspars Balodis, Jānis Iraids, Martins Kokainis, Krišjānis Prūsis, and Jevgēnijs Vihrovs. "Quantum speedups for exponential-time dynamic programming algorithms". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2019, pp. 1783–1793.

[10] S. Borağan Aruoba and Jesús Fernández-Villaverde. "A comparison of programming languages in macroeconomics". In: *Journal of Economic Dynamics and Control* 58 (2015), pp. 265–273. ISSN: 0165-1889. DOI: 10.1016/j.jedc.2015.05.009.

[11] Frank Arute et al. "Quantum Supremacy using a Programmable Superconducting Processor". In: *Nature* 574 (2019), pp. 505–510. DOI: 10.1038/s41586-019-1666-5.

[12] Sahel Ashhab, Naoki Yamamoto, Fumiki Yoshihara, and Kouichi Semba. "Numerical analysis of quantum circuits for state preparation and unitary operator synthesis". In: *Phys. Rev. A* 106 (2 Aug. 2022), p. 022426. DOI: 10.1103/PhysRevA.106.022426.

[13] Abhishek Awasthi et al. “Quantum Computing Techniques for Multi-knapsack Problems”. In: *Lecture Notes in Networks and Systems*. Springer Nature Switzerland, 2023, pp. 264–284. DOI: 10.1007/978-3-031-37963-5_19.

[14] Ryan Babbush, Jarrod R. McClean, Michael Newman, Craig Gidney, Sergio Boixo, and Hartmut Neven. “Focus beyond Quadratic Speedups for Error-Corrected Quantum Advantage”. In: *PRX Quantum* 2 (1 Mar. 2021), p. 010103. DOI: 10.1103/PRXQuantum.2.010103.

[15] R.N. Bailey, K.M. Garner, and M.F. Hobbs. “Using simulated annealing and genetic algorithms to solve staff-scheduling problems”. In: *Asia-Pacific Journal of Operational Research* 14.2 (1997), p. 27.

[16] W. P. Banner, Shima Bab Hadiashar, Grzegorz Mazur, Tim Menke, Marcin Ziolkowski, Ken Kennedy, Jhonathan Romero, Yudong Cao, Jeffrey A. Grover, and William D. Oliver. *Quantum Inspired Optimization for Industrial Scale Problems*. 2023. arXiv: 2305.02179 [quant-ph].

[17] Francisco Barahona, Martin Grötschel, Michael Jünger, and Gerhard Reinelt. “An Application of Combinatorial Optimization to Statistical Physics and Circuit Layout Design”. In: *Oper. Res.* 36 (1988), pp. 493–513.

[18] Jonathan F. Bard, Canan Binici, and Anura H. deSilva. “Staff scheduling at the United States Postal Service”. In: *Computers & Operations Research* 30.5 (2003), pp. 745–771. ISSN: 0305-0548. DOI: 10.1016/S0305-0548(02)00048-5.

[19] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. “Elementary gates for quantum computation”. In: *Phys. Rev. A* 52 (5 Nov. 1995), pp. 3457–3467. DOI: 10.1103/PhysRevA.52.3457.

[20] Andreas Bayerstadler et al. “Industry quantum computing applications”. In: *EPJ Quantum Technology* 8.1 (Nov. 2021), p. 25. ISSN: 2196-0763. DOI: 10.1140/epjqt/s40507-021-00114-x.

[21] David Beckman, Amalavoyal N. Chari, Srikrishna Devabhaktuni, and John Preskill. “Efficient networks for quantum factoring”. In: *Physical Review A* 54.2 (Aug. 1996), pp. 1034–1063. ISSN: 1094-1622. DOI: 10.1103/physreva.54.1034.

[22] Michael E. Beverland, Prakash Murali, Matthias Troyer, Krysta M. Svore, Torsten Hoefler, Vadym Kliuchnikov, Guang Hao Low, Mathias Soeken, Aarthi Sundaram, and Alexander Vaschillo. *Assessing requirements to scale to practical quantum advantage*. 2022. arXiv: 2211.07629 [quant-ph].

[23] Susan Blackford, Ross Moore, and Nikos Drakos. *LAPACK Users' Guide*. Accessed on: 23-10-2020. URL: http://www.netlib.org/lapack/lug/node71.html.

[24] D. Bluvstein, Simon J Evered, Alexandra A Geim, Sophie H Li, Hengyun Zhou, Tom Manovitz, Sepehr Ebadi, Madelyn Cain, Marcin Kalinowski, Dominik Hangleiter, et al. “Logical quantum processor based on reconfigurable atom arrays”. In: *Nature* 626.7997 (2024).

[25] Sergio Boixo, Sergei V. Isakov, Vadim N. Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J. Bremner, John M. Martinis, and Hartmut Neven. "Characterizing quantum supremacy in near-term devices". In: *Nature Physics* 14.6 (Apr. 2018), pp. 595–600. ISSN: 1745-2481. DOI: 10.1038/s41567-018-0124-x.

[26] Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, and Paul Zimmermann. "Comparing the difficulty of factorization and discrete logarithm: a 240-digit experiment". In: *Advances in Cryptology – CRYPTO 2020*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12171. Lecture Notes in Computer Science. Santa Barbara CA, United States: Springer, Aug. 2020, pp. 62–91. DOI: 10.1007/978-3-030-56880-1_3. arXiv: 2006.06197.

[27] Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, and Paul Zimmermann. "The State of the Art in Integer Factoring and Breaking Public-Key Cryptography". In: *IEEE Security & Privacy* 20.2 (2022), pp. 80–86. DOI: 10.1109/MSEC.2022.3141918.

[28] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. "Tight Bounds on Quantum Searching". In: *Fortschritte der Physik* 46.4-5 (June 1998), pp. 493–505. DOI: 10.1002/(sici)1521-3978(199806)46:4/5<493::aid-prop493>3.0.co;2-p.

[29] S. Bravyi and A. Kitaev. "Universal quantum computation with ideal Clifford gates and noisy ancillas". In: *Physical Review A* 71.2 (Feb. 2005). ISSN: 1094-1622. DOI: 10.1103/physreva.71.022316.

[30] Sergey Bravyi, Andrew W. Cross, Jay M. Gambetta, Dmitri Maslov, Patrick Rall, and Theodore J. Yoder. "High-threshold and low-overhead fault-tolerant quantum memory". In: *Nature* 627.8005 (Mar. 2024), pp. 778–782. ISSN: 1476-4687. DOI: 10.1038/s41586-024-07107-7.

[31] David Bulger, William P Baritompa, Graham R Wood, et al. "Implementing pure adaptive search with Grover's quantum algorithm". In: *Journal of optimization theory and applications* 116.3 (2003), pp. 517–529.

[32] Stephen S. Bullock and Igor L. Markov. "Arbitrary two-qubit computation in 23 elementary gates". In: *Physical Review A* 68.1 (July 2003). ISSN: 1094-1622. DOI: 10.1103/physreva.68.012318.

[33] J. M. Cargal. "Chapter 31: Geometric series". In: *Discrete Mathematics for Neophytes: Number Theory, Probability, Algorithms, and Other Stuff*. 1991. URL: https://www.cargalmathbooks.com/lectures.htm.

[34] Davide Castelvecchi. "IBM releases first-ever 1,000-qubit quantum chip". In: *Nature* 624.238 (2023). DOI: 10.1038/d41586-023-03854-1.

[35] Liangyu Chen et al. "Transmon qubit readout fidelity at the threshold for quantum error correction without a quantum-limited amplifier". In: *npj Quantum Information* 9.1 (Mar. 2023). DOI: 10.1038/s41534-023-00689-6.

[36] Tim Chen and Stefan Natu. “Speeding up hybrid quantum algorithms with parametric circuits on Amazon Braket”. In: *AWS Quantum Technologies Blog* (Oct. 2023). URL: https://aws.amazon.com/blogs/quantum-computing/speeding-up-hybrid-quantum-algorithms-with-parametric-circuits-on-amazon-braket/ (visited on 08/19/2024).

[37] Frederic Chong and Margaret Martonosi. “Programming languages and compiler design for realistic quantum hardware”. In: *Nature* 549 (Sept. 2017), pp. 180–187. DOI: 10.1038/nature23459.

[38] Atom Computing. “Atom Computing Unveils First-Gen Quantum Computing System, Closes $15M Series A”. In: *hpcwire.com* (July 2021). URL: www.hpcwire.com/off-the-wire/atom-computing-unveils-first-gen-quantum-computing-system-closes-15m-series-a/ (visited on 01/24/2024).

[39] Rigetti Computing. *PyQuil documention*. [Accessed: Aug. 4, 2023]. 2021. URL: https://pyquil-docs.rigetti.com/en/stable.

[40] Rigetti Computing. “Rigetti Announces Public Availability of Ankaa-2 System with a 2.5x Performance Improvement Compared to Previous QPUs”. In: *GlobeNewswire.com* (Jan. 2024). URL: www.globenewswire.com/news-release/2024/01/04/2804006/0/en/Rigetti-Announces-Public-Availability-of-Ankaa-2-System-with-a-2-5x-Performance-Improvement-Compared-to-Previous-QPUs.html (visited on 01/24/2024).

[41] Rigetti Computing. *Scalable quantum systems built from the chip up to power practical applications*. URL: www.rigetti.com/what-we-build (visited on 01/24/2024).

[42] Standard Performance Evaluation Corporation. *SPEC Benchmark Results for SPECint 2017, 2006, 2000, 95 and SPECpower_ssj 2008*. (1991–2024). URL: https://www.spec.org/ (visited on 08/19/2024).

[43] James R. Cruise, Neil I. Gillespie, and Brendan Reid. *Practical Quantum Computing: The value of local computation*. 2020. arXiv: 2009.08513 [quant-ph].

[44] G. Cybenko. “Reducing quantum computations to elementary unitary operations”. In: *Computing in Science & Engineering* 3.2 (2001), pp. 27–32. DOI: 10.1109/5992.908999.

[45] W. van Dam, M. Mykhailova, and M. Soeken. “Using Azure Quantum Resource Estimator for Assessing Performance of Fault Tolerant Quantum Computation”. In: *Proceedings of the SC'23 Workshops of The International Conference on HPC, Network, Storage, and Analysis*. New York, NY, USA: Association for Computing Machinery, 2023, pp. 1414–1419. DOI: 10.1145/3624062.3624211.

[46] James Dargan. “Quantum Journey From the Search Engine to Google Sycamore”. In: *thequantuminsider.com* (July 2022). URL: thequantuminsider.com/2022/07/14/google-sycamore/ (visited on 01/24/2024).

[47] Robert Davis. “What is quantum utility?” In: *ibm.com* (Nov. 2023). URL: https://www.ibm.com/quantum/blog/what-is-quantum-utlity (visited on 07/02/2024).

[48] C. M. Dawson and M. A. Nielsen. *The Solovay-Kitaev algorithm*. 2005. arXiv: quant-ph/0505030 [quant-ph].

[49] Alexis De Vos and Stijn De Baerdemacker. "A Unified Approach to Quantum Computation and Classical Reversible Computation". In: *Reversible Computation*. Ed. by Jarkko Kari and Irek Ulidowski. Cham: Springer International Publishing, 2018, pp. 133–143. ISBN: 978-3-319-99498-7.

[50] Alexis De Vos and Stijn De Baerdemacker. "Block-ZXZ synthesis of an arbitrary quantum circuit". In: *Phys. Rev. A* 94 (5 Nov. 2016), p. 052317. DOI: 10.1103/PhysRevA.94.052317.

[51] Henning Dekant, Henry Tregillus, Robert Tucci, and Tao Yin. *Qubiter at GitHub*. Accessed on: 23-10-2020. 2020. URL: https://github.com/artiste-qb-net/qubiter.

[52] David Elieser Deutsch and Roger Penrose. "Quantum computational networks". In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 425.1868 (1989), pp. 73–90. DOI: 10.1098/rspa.1989.0099.

[53] S. J. Devitt, W. J. Munro, and K. Nemoto. "Quantum error correction for beginners". In: *Reports on Progress in Physics* 76.7 (June 2013), p. 076001.

[54] David P. DiVincenzo. "The Physical Implementation of Quantum Computation". In: *Fortschritte der Physik* 48.9–11 (Sept. 2000), pp. 771–783. ISSN: 1521-3978. DOI: 10.1002/1521-3978(200009)48:9/11<771::aid-prop771>3.0.co;2-e.

[55] Kathryn A. Dowsland. "Nurse scheduling with tabu search and strategic oscillation". In: *European Journal of Operational Research* 106.2 (1998), pp. 393–407. ISSN: 0377-2217. DOI: 10.1016/S0377-2217(97)00281-6.

[56] Thomas G. Draper. *Addition on a Quantum Computer*. 2000. arXiv: quant-ph/0008033 [quant-ph].

[57] Suguru Endo, Zhenyu Cai, Simon C. Benjamin, and Xiao Yuan. "Hybrid Quantum-Classical Algorithms and Quantum Error Mitigation". In: *Journal of the Physical Society of Japan* 90.3 (2021), p. 032001. DOI: 10.7566/JPSJ.90.032001.

[58] A. Erhard et al. "Entangling logical qubits with lattice surgery". In: *Nature* 589.7841 (2021).

[59] Ameneh Farahani and Hamid Tohidi. "Integrated optimization of quality and maintenance: A literature review". In: *Computers & Industrial Engineering* 151 (2021), p. 106924. ISSN: 0360-8352. DOI: 10.1016/j.cie.2020.106924.

[60] Jernej Rudi Finzgar, Philipp Ross, Leonhard Holscher, Johannes Klepsch, and Andre Luckow. "QUARK: A Framework for Quantum Computing Application Benchmarking". In: *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, Sept. 2022. DOI: 10.1109/qce53715.2022.00042.

[61] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. "Surface codes: Towards practical large-scale quantum computation". In: *Physical Review A* 86.3 (Sept. 2012). ISSN: 1094-1622. DOI: 10.1103/physreva.86.032324.

[62] Hartmut Führ and Ziemowit Rzeszotnik. *On biunimodular vectors for unitary matrices*. 2015. arXiv: 1506.06738 [math.RT].

[63] Daniel J. Garcia and Fengqi You. "Supply chain design and optimization: Challenges and opportunities". In: *Computers & Chemical Engineering* 81 (2015). Special Issue: Selected papers from the 8th International Symposium on the Foundations of Computer-Aided Process Design (FOCAPD 2014), July 13-17, 2014, Cle Elum, Washington, USA, pp. 153–170. ISSN: 0098-1354. DOI: 10.1016/j.compchemeng.2015.03.015.

[64] Craig Gidney and Martin Ekerå. "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits". In: *Quantum* 5 (Apr. 2021), p. 433. ISSN: 2521-327X.

[65] Craig Gidney and Austin G. Fowler. "Efficient magic state factories with a catalyzed |CCZ⟩ to 2|T⟩ transformation". In: *Quantum* 3 (Apr. 2019), p. 135. ISSN: 2521-327X. DOI: 10.22331/q-2019-04-30-135.

[66] Austin Gilliam, Stefan Woerner, and Constantin Gonciulea. "Grover Adaptive Search for Constrained Polynomial Binary Optimization". In: *Quantum* 5 (Apr. 2021), p. 428. DOI: 10.22331/q-2021-04-08-428.

[67] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. "Quantum Random Access Memory". In: *Phys. Rev. Lett.* 100 (16 Apr. 2008), p. 160501. DOI: 10.1103/PhysRevLett.100.160501.

[68] "Glossary: Quantum Advantage". In: *quera.com* (). URL: https://www.quera.com/glossary/advantage (visited on 07/02/2024).

[69] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. 4th ed. The John Hopkins University Press. ISBN: 9781421407944. DOI: 10.56021/9781421407944.

[70] Daniel Gottesman. "An introduction to quantum error correction and fault-tolerant quantum computation". In: *Quantum information science and its contributions to mathematics, Proceedings of Symposia in Applied Mathematics*. Vol. 68. 2010, pp. 13–58.

[71] Frank Gray. *Pulse code communication*. U.S. Patent no. 2,632,058. Mar. 1953.

[72] Harper R. Grimsley, Sophia E. Economou, Edwin Barnes, and Nicholas J. Mayhall. "An adaptive variational algorithm for exact molecular simulations on a quantum computer". In: *Nature Communications* 10.1 (July 2019). ISSN: 2041-1723. DOI: 10.1038/s41467-019-10988-2.

[73] Lov 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. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 212–219. ISBN: 0897917855. DOI: 10.1145/237814.237866.

[74] Lov K. Grover. "Quantum Mechanics Helps in Searching for a Needle in a Haystack". In: *Phys. Rev. Lett.* 79 (2 July 1997), pp. 325–328. DOI: 10.1103/PhysRevLett.79.325.

[75] Alexandre Guillaume, Edwin Y. Goh, Mark D. Johnston, Brian D. Wilson, Anita Ramanan, Frances Tibble, and Brad Lackey. "Deep Space Network Scheduling Using Quantum Annealing". In: *IEEE Transactions on Quantum Engineering* 3 (2022), pp. 1–13. DOI: 10.1109/TQE.2022.3199267.

[76] Hubert de Guise, Olivia Di Matteo, and Luis L. Sánchez-Soto. "Simple factorization of unitary transformations". In: *Physical Review A* 97.2 (Feb. 2018). ISSN: 2469-9934. DOI: 10.1103/physreva.97.022328.

[77] Walter J. Gutjahr and Marion S. Rauner. "An ACO algorithm for a dynamic regional nurse-scheduling problem in Austria". In: *Computers & Operations Research* 34.3 (2007). Logistics of Health Care Management, pp. 642–666. ISSN: 0305-0548. DOI: 10.1016/j.cor.2005.03.018.

[78] H. Haffner, C. Roos, and R. Blatt. "Quantum computing with trapped ions". In: *Physics Reports* 469.4 (Dec. 2008), pp. 155–203. DOI: 10.1016/j.physrep.2008.09.003.

[79] M Hafizi, S N S Jamaludin, and A H Shamil. "State of The Art Review of Quality Control Method in Automotive Manufacturing Industry". In: 530.1 (June 2019), p. 012034. DOI: 10.1088/1757-899X/530/1/012034.

[80] Wei Hai-Rui, Di Yao-Min, and Zhang-Jie. "Modified Khaneja-Glaser Decomposition and Realization of Three-Qubit Quantum Gate". In: *Chinese Physics Letters* 25.9 (Sept. 2008), p. 3107. DOI: 10.1088/0256-307X/25/9/004.

[81] E. Hansen, S. Joshi, and H. Rarick. "Resource Estimation of Quantum Multiplication Algorithms". In: *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Vol. 2. IEEE. 2023, pp. 199–202.

[82] Ian Hellström. *Quantum Computer Roadmaps*. URL: https://ianhellstrom.org/quantum.html (visited on 07/10/2024).

[83] B. Hetényi and J. R. Wootton. *Creating entangled logical qubits in the heavy-hex lattice with topological codes*. 2024. arXiv: 2404.15989 [quant-ph].

[84] Torsten Hoefler, Thomas Haener, and Matthias Troyer. *Disentangling Hype from Practicality: On Realistically Achieving Quantum Advantage*. 2023. arXiv: 2307.00523 [quant-ph].

[85] Ernst Joachim Houtgast, Vlad-Mihai Sima, Koen Bertels, and Zaid Al-Ars. "Hardware acceleration of BWA-MEM genomic short read mapping for longer read lengths". In: *Computational Biology and Chemistry* 75 (2018), pp. 54–64. ISSN: 1476-9271. DOI: 10.1016/j.compbiolchem.2018.03.024.

[86] Jeremy Hsu. "CES 2018: Intel's 49-Qubit Chip Shoots for Quantum Supremacy". In: *IEEE Spectrum* (Jan. 2018). URL: spectrum.ieee.org/intels-49qubit-chip-aims-for-quantum-supremacy (visited on 01/24/2024).

[87] Travis Humble, Alexander McCaskey, Dmitry Lyakh, Meenambika Gowrishankar, Albert Frisch, and Thomas Monz. "Quantum Computers for High-Performance Computing". In: *IEEE Micro* 41 (Sept. 2021), pp. 15–23. DOI: 10.1109/MM.2021.3099140.

[88] Kazuki Ikeda, Yuma Nakamura, and Travis S. Humble. “Application of Quantum Annealing to Nurse Scheduling Problem”. In: *Scientific Reports* 9.1 (Sept. 2019). DOI: 10.1038/s41598-019-49172-3.

[89] Wafer World Inc. “How Small Can Transistors Get?” In: *waferworld.com* (Nov. 2021). URL: https://www.waferworld.com/post/how-small-can-transistors-get (visited on 08/15/2024).

[90] QuEra Computing Inc. “Qauntum Noise”. In: *QuEra Glossary* (2023). URL: https://www.quera.com/glossary/noise (visited on 08/15/2024).

[91] IonQ. “IonQ Aria: Practical Performance”. In: *ionq.com* (Jan. 2024). URL: ionq.com/resources/ionq-aria-practical-performance (visited on 01/24/2024).

[92] Raban Iten, Roger Colbeck, Ivan Kukuljan, Jonathan Home, and Matthias Christandl. “Quantum circuits for isometries”. In: *Physical Review A* 93.3 (Mar. 2016). ISSN: 2469-9934. DOI: 10.1103/physreva.93.032318.

[93] Raban Iten, Oliver Reardon-Smith, Emanuel Malvetti, Luca Mondada, Gabrielle Pauvert, Ethan Redmond, Ravjot Singh Kohli, and Roger Colbeck. *Introduction to UniversalQCompiler*. 2021. arXiv: 1904.01072 [quant-ph].

[94] Gui-Long Jiang, Wen-Qiang Liu, and Hai-Rui Wei. “Optimal Quantum Circuits for General Multi-Qutrit Quantum Computation”. In: *Adv. Quantum Technol.* 7.7 (2024), p. 2400033. DOI: 10.1002/qute.202400033. arXiv: 2310.11996 [quant-ph].

[95] Cody Jones. “Low-overhead constructions for the fault-tolerant Toffoli gate”. In: *Phys. Rev. A* 87 (2 Feb. 2013), p. 022328. DOI: 10.1103/PhysRevA.87.022328.

[96] Petar Jurcevic et al. “Demonstration of quantum volume 64 on a superconducting quantum computing system”. In: *Quantum Science and Technology* 6.2 (Mar. 2021), p. 025020. ISSN: 2058-9565. DOI: 10.1088/2058-9565/abe519.

[97] Özgür Kabak, Füsun Ülengin, Emel Aktaş, Şule Önsel, and Y. Ilker Topcu. “Efficient shift scheduling in the retail sector through two-stage optimization”. In: *European Journal of Operational Research* 184.1 (2008), pp. 76–90. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2006.10.039.

[98] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M. Chow, and Jay M. Gambetta. “Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets”. In: *Nature* 549.7671 (Sept. 2017), pp. 242–246. ISSN: 1476-4687. DOI: 10.1038/nature23879.

[99] Peter J. Karalekas, Nikolas A. Tezak, Eric C. Peterson, Colm A. Ryan, Marcus P. da Silva, and Robert S. Smith. “A quantum-classical cloud platform optimized for variational hybrid algorithms”. In: *Quantum Science and Technology* 5.2 (Apr. 2020), p. 024003. DOI: 10.1088/2058-9565/ab7559.

[100] Richard M. Karp. “Reducibility among Combinatorial Problems”. In: *Complexity of Computer Computations*. Ed. by Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger. Boston, MA: Springer US, 1972, pp. 85–103. ISBN: 978-1-4684-2001-2. DOI: 10.1007/978-1-4684-2001-2_9.

[101] Nader Khammassi, Imran Ashraf, Xiang Fu, Carmina García Almudever, and Koen Bertels. “QX: A high-performance quantum computer simulation platform”. English. In: *Proceedings of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Design, Automation and Test in Europe : DATE 17 ; Conference date: 27-03-2017 Through 31-03-2017. United States: IEEE, 2017, pp. 464–469. ISBN: 978-1-5090-5826-6. DOI: 10.23919/DATE.2017.7927034.

[102] Nader Khammassi, Imran Ashraf, Adriaan Rol, Xiang Fu, Wouter Vlothuizen, Hans van Someren, and more. *OpenQL documentation*. [Accessed: Aug. 4, 2023]. URL: https://openql.readthedocs.io/.

[103] Nader Khammassi, Imran Ashraf, J. V. Someren, R. Nane, A. M. Krol, M. A. Rol, L. Lao, K. Bertels, and C. G. Almudever. “OpenQL: A Portable Quantum Programming Framework for Quantum Accelerators”. In: *J. Emerg. Technol. Comput. Syst.* 18.1 (Dec. 2021). ISSN: 1550-4832. DOI: 10.1145/3474222.

[104] Navin Khaneja and Steffen J. Glaser. “Cartan decomposition of SU(2n) and control of spin systems”. In: *Chemical Physics* 267.1 (2001), pp. 11–23. ISSN: 0301-0104. DOI: 10.1016/S0301-0104(01)00318-4.

[105] Min-Soo Kim, Seog-Chan Oh, Eun Hyo Chang, Sangheon Lee, James W. Wells, Jorge Arinez, and Young Jae Jang. “A dynamic programming-based heuristic algorithm for a flexible job shop scheduling problem of a matrix system in automotive industry”. In: *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*. 2022, pp. 777–782. DOI: 10.1109/CASE49997.2022.9926440.

[106] Youngseok Kim et al. “Evidence for the utility of quantum computing before fault tolerance”. In: *Nature* 618 (June 2023), pp. 500–505. DOI: 10.1038/s41586-023-06096-3.

[107] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. “Optimization by Simulated Annealing”. In: *Science* 220.4598 (1983), pp. 671–680.

[108] A.Y. Kitaev. “Fault-tolerant quantum computation by anyons”. In: *Annals of Physics* 303.1 (2003), pp. 2–30. ISSN: 0003-4916. DOI: 10.1016/S0003-4916(02)00018-0.

[109] A.Y. Kitaev. “Quantum computations: algorithms and error correction”. In: *Russian Mathematical Surveys* 52.6 (Dec. 1997), p. 1191. DOI: 10.1070/RM1997v052n06ABEH002155.

[110] Morten Kjaergaard, Mollie E. Schwartz, Jochen Braumüller, Philip Krantz, Joel I.-J. Wang, Simon Gustavsson, and William D. Oliver. “Superconducting Qubits: Current State of Play”. In: *Annual Review of Condensed Matter Physics* 11.Volume 11, 2020 (2020), pp. 369–395. ISSN: 1947-5462. DOI: 10.1146/annurev-conmatphys-031119-050605.

[111] Vadym Kliuchnikov, Kristin Lauter, Romy Minko, Adam Paetznick, and Christophe Petit. "Shorter quantum circuits via single-qubit gate approximation". In: *Quantum* 7 (Dec. 2023), p. 1208. ISSN: 2521-327X. DOI: 10.22331/q-2023-12-18-1208. URL: https://doi.org/10.22331/q-2023-12-18-1208.

[112] E. Knill. *Approximation by Quantum Circuits*. 1995. arXiv: quant-ph/9508006 [quant-ph].

[113] S. Krinner et al. "Realizing repeated quantum error correction in a distance-three surface code". In: *Nature* 605.7911 (2022).

[114] Anna M. Krol and Zaid Al-Ars. "Beyond quantum Shannon decomposition: Circuit construction for n-qubit gates based on block-ZXZ decomposition". In: *Phys. Rev. Appl.* 22 (3 Sept. 2024), p. 034019. DOI: 10.1103/PhysRevApplied.22.034019.

[115] Anna M. Krol, Marvin Erdmann, Rajesh Mishra, Phattharaporn Singkanipa, Ewan Munro, Marcin Ziolkowski, Andre Luckow, and Zaid Al-Ars. *QISS: Quantum Industrial Shift Scheduling Algorithm*. Submitted to *IEEE Transactions on Quantum Engineering*. 2024. arXiv: 2401.07763 [quant-ph].

[116] Anna M. Krol, Marvin Erdmann, Ewan Munro, Andre Luckow, and Zaid Al-Ars. "Assessing the Requirements for Industry Relevant Quantum Computation". In: *Proceedings of the 2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Montréal, Canada, 2024. arXiv: 2408.02587 [quant-ph].

[117] Anna M. Krol, Koen Mesman, Aritra Sarkar, and Zaid Al-Ars. "Efficient Parameterised Compilation for Hybrid Quantum Programming". In: *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Vol. 2. 2023, pp. 103–111. DOI: 10.1109/QCE57702.2023.10192.

[118] Anna M. Krol, Aritra Sarkar, Imran Ashraf, Zaid Al-Ars, and Koen Bertels. "Efficient Decomposition of Unitary Matrices in Quantum Circuit Compilers". In: *Applied Sciences* 12.2 (2022). ISSN: 2076-3417. DOI: 10.3390/app12020759.

[119] Benjamin P Lanyon, Marco Barbieri, Marcelo P Almeida, Thomas Jennewein, Timothy C Ralph, Kevin J Resch, Geoff J Pryde, Jeremy L O'brien, Alexei Gilchrist, and Andrew G White. "Simplifying quantum logic using higher-dimensional Hilbert spaces". In: *Nature Physics* 5.2 (2009), pp. 134–140. DOI: 10.1038/nphys1150.

[120] Steven Leibson. "Physicists Pushing Boundaries Of Physics Using Quantum Computers". In: *Forbes.com* (Aug. 2023). URL: www.forbes.com/sites/tiriasresearch/2023/08/03/physicists-pushing-boundaries-of-physics-using-quantum-computers/ (visited on 01/24/2024).

[121] Charles E. Leiserson, Neil C. Thompson, Joel S. Emer, Bradley C. Kuszmaul, Butler W. Lampson, Daniel Sanchez, and Tao B. Schardl. "There's plenty of room at the Top: What will drive computer performance after Moore's law?" In: *Science* 368.6495 (2020), eaam9744. DOI: 10.1126/science.aam9744. eprint: https://www.science.org/doi/pdf/10.1126/science.aam9744.

[122] Taiwan Semiconductor Manufacturing Company Limited. “3nm Technology”. In: *tsmc.com* (2022). URL: https://www.tsmc.com/english/dedicatedFoundry/technology/logic/l_3nm (visited on 08/15/2024).

[123] Norbert M. Linke, Dmitri Maslov, Martin Roetteler, Shantanu Debnath, Caroline Figgatt, Kevin A. Landsman, Kenneth Wright, and Christopher Monroe. “Experimental comparison of two quantum computing architectures”. In: *Proceedings of the National Academy of Sciences* 114.13 (2017), pp. 3305–3310. DOI: 10.1073/pnas.1618020114.

[124] Jin-Guo Liu, Yi-Hong Zhang, Yuan Wan, and Lei Wang. “Variational quantum eigensolver with fewer qubits”. In: *Physical Review Research* 1.2 (Sept. 2019). ISSN: 2643-1564. DOI: 10.1103/physrevresearch.1.023025.

[125] Sonia Lopez, Bradben, et al. *Azure Quantum documentation (preview)*. Microsoft 2023. URL: https://learn.microsoft.com/en-us/azure/quantum/ (visited on 07/2023).

[126] Andre Luckow, Johannes Klepsch, and Josef Pichlmeier. “Quantum Computing: Towards Industry Reference Problems”. In: *Digitale Welt* 5.2 (Mar. 2021), pp. 38–45. ISSN: 2569-1996. DOI: 10.1007/s42354-021-0335-7.

[127] Quantum Machines. *Quantum Machines*. [Accessed: Aug. 4, 2023]. URL: https://www.quantum-machines.co.

[128] Lars S Madsen, Fabian Laudenbach, Mohsen Falamarzi Askarani, Fabien Rortais, Trevor Vincent, Jacob FF Bulmer, Filippo M Miatto, Leonhard Neuhaus, Lukas G Helt, Matthew J Collins, et al. “Quantum computational advantage with a programmable photonic processor”. In: *Nature* 606.7912 (2022), pp. 75–81.

[129] Maximilian Balthasar Mansky, Santiago Londoño Castillo, Victor Ramos Puigvert, and Claudia Linnhoff-Popien. “Near-optimal quantum circuit construction via Cartan decomposition”. In: *Phys. Rev. A* 108 (5 Nov. 2023), p. 052607. DOI: 10.1103/PhysRevA.108.052607.

[130] Sara Mattia, Fabrizio Rossi, Mara Servilio, and Stefano Smriglio. “Staffing and scheduling flexible call centers by two-stage robust optimization”. In: *Omega* 72 (2017), pp. 25–37. ISSN: 0305-0483. DOI: 10.1016/j.omega.2016.11.001.

[131] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. “The theory of variational hybrid quantum-classical algorithms”. In: *New Journal of Physics* 18.2 (Feb. 2016), p. 023023. DOI: 10.1088/1367-2630/18/2/023023.

[132] Jim McGregor. “Intel Tunnel Falls Into Quantum Computing”. In: *Forbes.com* (June 2023). URL: www.forbes.com/sites/tiriasresearch/2023/06/15/intel-tunnel-falls-into-quantum-computing/ (visited on 01/24/2024).

[133] Koen Mesman, Zaid Al-Ars, and Matthias Möller. “QPack: Quantum Approximate Optimization Algorithms as universal benchmark for quantum computers”. In: (2022). arXiv: 2103.17193 [cs.ET].

[134] Adam R. Mills, Charles R. Guinn, Michael J. Gullans, Anthony J. Sigillito, Mayer M. Feldman, Erik Nielsen, and Jason R. Petta. "Two-qubit silicon quantum processor with operation fidelity exceeding 99%". In: *Science Advances* 8.14 (Apr. 2022). DOI: oi.org/10.1126/sciadv.abn5130.

[135] Matthias Möller and Cornelis Vuik. "On the impact of quantum computing technology on future developments in high-performance scientific computing". In: *Ethics and Information Technology* 19 (2017), pp. 253–269. URL: https://api.semanticscholar.org/CorpusID:6864503.

[136] Ashley Montanaro. "Quantum speedup of branch-and-bound algorithms". In: *Phys. Rev. Res.* 2 (1 Jan. 2020), p. 013056. DOI: 10.1103/PhysRevResearch.2.013056.

[137] Gary J. Mooney, Charles D. Hill, and Lloyd C. L. Hollenberg. "Entanglement in a 20-Qubit Superconducting Quantum Computer". In: *Scientific Reports* 9.1 (Sept. 2019). ISSN: 2045-2322. DOI: 10.1038/s41598-019-49805-7.

[138] Patrick Moorhead. "Xanadu Brings Photonic Quantum Computing To The Cloud". In: *Moor Insights & Strategy* (Sept. 2020). URL: moorinsightsstrategy.com/xanadu-brings-photonic-quantum-computing-to-the-cloud/ (visited on 01/24/2024).

[139] Mikko Möttönen and Juha J. Vartiainen. "Decompositions of general quantum gates". In: *Trends in quantum computing research*. Ed. by Susan Shannon. Nova Science Publishers, 2006. Chap. 7, pp. 149–170. ISBN: 9781594548406.

[140] Mikko Möttönen, Juha J. Vartiainen, Ville Bergholm, and Martti M. Salomaa. "Quantum Circuits for General Multiqubit Gates". In: *Phys. Rev. Lett.* 93 (13 Sept. 2004), p. 130502. DOI: 10.1103/PhysRevLett.93.130502.

[141] Ken M. Nakanishi, Takahiko Satoh, and Synge Todo. *Quantum-gate decomposer*. 2021. arXiv: 2109.13223 [quant-ph].

[142] H. Neven and J. Kelly. *Suppressing quantum errors by scaling a surface code logical qubit*. 2023. URL: https://research.google/blog/suppressing-quantum-errors-by-scaling-a-surface-code-logical-qubit/ (visited on 07/01/2024).

[143] Jordan Novet. "Intel shows off its latest chip for quantum computing as it looks past Moore's Law". In: *CNBC.com* (Oct. 2017). URL: www.cnbc.com/2017/10/10/intel-delivers-17-qubit-quantum-computing-chip-to-qutech.html (visited on 01/24/2024).

[144] Dan O'Shea. *Rigetti details new QPU roadmap, fab expansion, Bluefors partnership*. URL: www.insidequantumtechnology.com/news-archive/rigetti-details-new-qpu-roadmap-fab-expansion-bluefors-partnership/ (visited on 01/24/2024).

[145] OQC. "OQC Honors Physicist Toshiko Yuasa with OQC Toshiko Platform, Secures $100M Funding". In: *hpcwire.com* (Nov. 2023). URL: www.hpcwire.com/off-the-wire/oqc-honors-physicist-toshiko-yuasa-with-oqc-toshiko-platform-secures-100m-funding/ (visited on 01/24/2024).

[146] Emir Hüseyin Özder, Evrencan Özcan, and Tamer Eren. "A Systematic Literature Review for Personnel Scheduling Problems". In: *International Journal of Information Technology & Decision Making* 19.06 (2020), pp. 1695–1735. DOI: 10.1142/S0219622020300050.

[147] C.C. Paige and M. Wei. "History and generality of the CS decomposition". In: *Linear Algebra and its Applications* 208-209 (1994), pp. 303–326. ISSN: 0024-3795. DOI: 10.1016/0024-3795(94)90446-4.

[148] Aidan Pellow-Jarman, Ilya Sinayskiy, Anban Pillay, and Francesco Petruccione. "A comparison of various classical optimizers for a variational quantum linear solver". In: *Quantum Information Processing* 20.6 (June 2021). DOI: 10.1007/s11128-021-03140-x.

[149] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. "A variational eigenvalue solver on a photonic quantum processor". In: *Nature Communications* 5.1 (July 2014). ISSN: 2041-1723. DOI: 10.1038/ncomms5213.

[150] Rebecca Pool. "A new kind of quantum". In: *SPIE.org* (Nov. 2020). URL: spie.org/news/photonics-focus/novdec-2020/a-new-kind-of-quantum (visited on 01/24/2024).

[151] L. Postler et al. "Demonstration of fault-tolerant universal quantum gate operations". In: *Nature* 605.7911 (2022).

[152] John Preskill. *Quantum computing and the entanglement frontier*. 2012. arXiv: 1203.5813 [quant-ph]. URL: https://arxiv.org/abs/1203.5813.

[153] Qiskit Development team. *Qiskit documentation*. [Accessed: Aug. 4, 2023]. 2020. URL: https://qiskit.org/documentation/.

[154] *Qualtran documentation*. Google LLC 2023. URL: https://qualtran.readthedocs.io/en/latest/index.html (visited on 11/2023).

[155] Quantinuum. "The Future of Quantum Hardware". In: (). URL: www.quantinuum.com/hardware (visited on 01/24/2024).

[156] Péter Rakyta and Zoltán Zimborás. "Approaching the theoretical limit in quantum gate decomposition". In: *Quantum* 6 (May 2022), p. 710. ISSN: 2521-327X. DOI: 10.22331/q-2022-05-11-710.

[157] Quantum Computing Report. "Atom Computing Previews an 1180 Qubit Neutral Atom Processor". In: *quantumcomputingreport.com* (Oct. 2023). URL: quantumcomputingreport.com/atom-computing-previews-an-1180-qubit-neutral-atom-processor (visited on 01/24/2024).

[158] Quantum Computing Report. *Zapata Introduces BenchQ, an Open-source Tool for Quantum Computing Resource Estimation and Benchmarking*. Dec. 2023. URL: https://quantumcomputingreport.com/zapata-introduces-benchq-an-open-source-tool-for-quantum-computing-resource-estimation-and-benchmarking/ (visited on 04/30/2024).

[159] Riverlane. *Deltaflow®*. [Accessed: Aug. 4, 2023]. URL: https://www.riverlane.com/products/.

[160] Marta Rocha, José Oliveira, and Maria Carravilla. "A constructive heuristic for staff scheduling in the glass industry". In: *Annals of Operations Research* 217.1 (June 2014), pp. 463–478. DOI: 10.1007/s10479-013-1525-y.

[161] J. Roffe. "Quantum error correction: an introductory guide". In: *Contemporary Physics* 60.3 (2019), pp. 226–245. DOI: 10.1080/00107514.2019.1667078.

[162] N. J. Ross and P. Selinger. *Optimal ancilla-free Clifford+T approximation of z-rotations*. 2016. arXiv: 1403.2975 [quant-ph].

[163] Karl Rupp. "40 Years of Microprocessor Trend Data". In: *karlrupp.net* (2018). URL: https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/ (visited on 08/19/2024).

[164] Karl Rupp. *Microprocessor Trend Data Repository*. 2022. URL: https://github.com/karlrupp/microprocessor-trend-data (visited on 08/19/2024).

[165] John Russell. "QuEra Debuts 3-Year Roadmap to 10,000 Physical and 100 Logical Qubits". In: *hpcwire.com* (Jan. 2024). URL: www.hpcwire.com/2024/01/09/quera-debuts-3-year-roadmap-to-10000-physical-and-100-logical-qubits/ (visited on 01/24/2024).

[166] C. Ryan-Anderson et al. "Realization of Real-Time Fault-Tolerant Quantum Error Correction". In: *Phys. Rev. X* 11 (4 Dec. 2021), p. 041058. DOI: 10.1103/PhysRevX.11.041058.

[167] Aritra Sarkar, Zaid Al-Ars, Carmen G. Almudever, and Koen L. M. Bertels. "QiBAM: Approximate Sub-String Index Search on Quantum Accelerators Applied to DNA Read Alignment". In: *Electronics* 10.19 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10192433.

[168] Aritra Sarkar, Zaid Al-Ars, and Koen Bertels. "QKSA: Quantum Knowledge Seeking Agent". In: *Artificial General Intelligence: 15th International Conference, AGI 2022, Seattle, WA, USA, August 19–22, 2022, Proceedings*. Seattle, WA, USA: Springer-Verlag, 2023, pp. 384–393. ISBN: 978-3-031-19906-6. DOI: 10.1007/978-3-031-19907-3_37.

[169] Aritra Sarkar, Zaid Al-Ars, and Koen Bertels. "QuASeR: Quantum Accelerated de novo DNA sequence reconstruction". In: *PLOS ONE* 16.4 (Apr. 2021), pp. 1–23. DOI: 10.1371/journal.pone.0249850.

[170] Alistair Savage. "Introduction to Lie Groups". In: *Course notes of* MAT1411/MAT5158. University of Ottawa, 2015. eprint: https://alistairsavage.ca/mat4144/notes/MAT4144-5158-LieGroups.pdf.

[171] Martin J.A. Schuetz, J. Kyle Brubaker, Henry Montagu, Yannick van Dijk, Johannes Klepsch, Philipp Ross, Andre Luckow, Mauricio G.C. Resende, and Helmut G. Katzgraber. "Optimization of Robot-Trajectory Planning with Nature-Inspired and Hybrid Quantum Algorithms". In: *Phys. Rev. Appl.* 18 (5 Nov. 2022), p. 054045. DOI: 10.1103/PhysRevApplied.18.054045.

[172] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. "Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures". In: *Proceedings of the 50th Annual Design Automation Conference*. DAC '13. Austin, Texas: Association for Computing Machinery, 2013. ISBN: 9781450320719. DOI: 10.1145/2463209.2488785.

[173] V.V. Shende, S.S. Bullock, and I.L. Markov. "Synthesis of quantum-logic circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25.6 (June 2006), pp. 1000–1010. ISSN: 1937-4151. DOI: 10.1109/tcad.2005.855930.

[174] V.V. Shende, I.L. Markov, and S.S. Bullock. "Smaller two-qubit circuits for quantum communication and computation". In: *Proceedings Design, Automation and Test in Europe Conference and Exhibition*. Vol. 2. 2004, 980–985 Vol.2. DOI: 10.1109/DATE.2004.1269020.

[175] Vivek V. Shende, Stephen S. Bullock, and Igor L. Markov. *A Practical Top-down Approach to Quantum Circuit Synthesis*. 2004. arXiv: quant-ph/0406176v3.

[176] Vivek V. Shende, Igor L. Markov, and Stephen S. Bullock. "Minimal universal two-qubit controlled-NOT-based circuits". In: *Physical Review A* 69.6 (June 2004). ISSN: 1094-1622. DOI: 10.1103/physreva.69.062321.

[177] Peter W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[178] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM J. Comput.* 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 0097-5397. DOI: 10.1137/S0097539795293172.

[179] Robert Smith, Will Zeng, Spike Curtis, Nick Rubin, Anthony Polloreno, Peter Karalekas, Nikolas Tezak, Chris Osborn, and many more. *Pyquil: Quantum programming in Python*. [Accessed: Aug. 4, 2023]. 2020. URL: https://github.com/rigetti/pyquil.

[180] Robert S. Smith, Michael J. Curtis, and William J. Zeng. *A Practical Quantum Instruction Set Architecture*. 2016. arXiv: 1608.03355 [quant-ph].

[181] Paul Smith-Goodson. "A Quantum Leap In AI: IonQ Aims To Create Quantum Machine Learning Models At The Level Of General Human Intelligence". In: *Forbes.com* (June 2023). URL: www.forbes.com/sites/moorinsights/2023/06/02/a-quantum-leap-in-ai-ionq-aims-to-create-quantum-machine-learning-models-at-the-level-of-general-human-intelligence/ (visited on 01/24/2024).

[182] P. B. M. Sousa and R. V. Ramos. *Universal quantum circuit for n-qubit quantum gate: A programmable quantum gate*. 2006. arXiv: quant-ph/0602174 [quant-ph].

[183] Annalise Stockley and Keith Briggs. "Optimizing antenna beamforming with quantum computing". In: *2023 17th European Conference on Antennas and Propagation (EuCAP)*. 2023, pp. 1–5. DOI: 10.23919/EuCAP57121.2023.10133700.

[184] Brian D. Sutton. *Computing the complete CS decomposition.* 2008. arXiv: 0707.1838 [math.NA].

[185] Matt Swayne. "OQC's 'Lucy' Becomes First European Quantum Computer on Amazon Braket". In: *thequantuminsider.com* (Feb. 2022). URL: thequantuminsider.com/2022/02/28/oqcs-lucy-becomes-first-european-quantum-computer-on-amazon-braket/ (visited on 01/24/2024).

[186] Dan Swinhoe. "IonQ announces two rack-mounted quantum computers". In: *datacenterdynamics.com* (Sept. 2023). URL: www.datacenterdynamics.com/en/news/ionq-announces-two-rack-mounted-quantum-computers/ (visited on 01/24/2024).

[187] Dan Swinhoe. *Quantinuum upgrades H2 quantum computer from 32 to 56 qubits.* June 2024. URL: https://www.datacenterdynamics.com/en/news/quantinuum-upgrades-h2-quantum-computer-from-32-to-56-qubits/ (visited on 07/01/2024).

[188] Swamit S. Tannu and Moinuddin K. Qureshi. "Not All Qubits Are Created Equal: A Case for Variability-Aware Policies for NISQ-Era Quantum Computers". In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems.* ASPLOS '19. Providence, RI, USA: Association for Computing Machinery, 2019, pp. 987–999. ISBN: 9781450362405. DOI: 10.1145/3297858.3304007.

[189] Jörn-Henrik Thun and Daniel Hoenig. "An empirical analysis of supply chain risk management in the German automotive industry". In: *International Journal of Production Economics* 131.1 (2011). Innsbruck 2008, pp. 242–249. ISSN: 0925-5273. DOI: 10.1016/j.ijpe.2009.10.010.

[190] John Timmer. "IBM releases 1,000+ qubit processor, roadmap to error correction". In: *arstechnica.com* (Apr. 2023). URL: https://arstechnica.com/science/2023/12/ibm-adds-error-correction-to-updated-quantum-computing-roadmap/ (visited on 01/24/2024).

[191] Robert R. Tucci. *A Rudimentary Quantum Compiler(2cnd Ed.)* 1999. arXiv: quant-ph/9902062 [quant-ph].

[192] Jorne Van den Bergh, Jeroen Beliën, Philippe De Bruecker, Erik Demeulemeester, and Liesje De Boeck. "Personnel scheduling: A literature review". In: *European Journal of Operational Research* 226.3 (2013), pp. 367–385. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2012.11.029.

[193] Rodney Van Meter and Dominic Horsman. "A blueprint for building a quantum computer". In: *Commun. ACM* 56.10 (Oct. 2013), pp. 84–93. ISSN: 0001-0782. DOI: 10.1145/2494568.

[194] Juha J. Vartiainen, Mikko Möttönen, and Martti M. Salomaa. "Efficient Decomposition of Quantum Gates". In: *Phys. Rev. Lett.* 92 (17 Apr. 2004), p. 177902. DOI: 10.1103/PhysRevLett.92.177902. arXiv: quant-ph/0312218 [quant-ph].

[195] Farrokh Vatan and Colin P. Williams. *Realization of a General Three-Qubit Quantum Gate.* Feb. 2004. arXiv: quant-ph/0401178 [quant-ph].

[196] Farrokh Vatan and Colin Williams. "Optimal quantum circuits for general two-qubit gates". In: *Phys. Rev. A* 69 (3 Mar. 2004), p. 032315. DOI: 10.1103/PhysRevA.69.032315.

[197] Dan Ventura and Tony Martinez. "Quantum associative memory". In: *Information Sciences* 124.1 (2000), pp. 273–296. ISSN: 0020-0255. DOI: 10.1016/S0020-0255(99)00101-2.

[198] G. Vidal and C. M. Dawson. "Universal quantum circuit for two-qubit transformations with three controlled-NOT gates". In: *Phys. Rev. A* 69 (1 Jan. 2004), 010301(R). DOI: 10.1103/PhysRevA.69.010301.

[199] D. S. Wang, A. G. Fowler, and L. C. L. Hollenberg. "Surface code quantum computing with error rates over 1%". In: *Phys. Rev. A* 83 (2 Feb. 2011), p. 020302. DOI: 10.1103/PhysRevA.83.020302.

[200] Rui-Sheng Wang and Li-Min Wang. "Maximum cut in fuzzy nature: Models and algorithms". In: *Journal of Computational and Applied Mathematics* 234.1 (2010), pp. 240–252. ISSN: 0377-0427. DOI: 10.1016/j.cam.2009.12.022.

[201] Wikipedia contributors. *Transistor count — Wikipedia, The Free Encyclopedia*. 2024. URL: https://en.wikipedia.org/w/index.php?title=Transistor%5C_count&oldid=1237724570 (visited on 08/19/2024).

[202] Wallace Witkowski. "'Moore's Law's dead,' Nvidia CEO Jensen Huang says in justifying gaming-card price hike". In: *MarketWatch* (Sept. 2022). URL: https://www.marketwatch.com/story/moores-laws-dead-nvidia-ceo-jensen-says-in-justifying-gaming-card-price-hike-11663798618 (visited on 08/15/2024).

[203] Jonathan Wurtz et al. *Aquila: QuEra's 256-qubit neutral-atom quantum computer*. 2023. arXiv: 2306.11727 [quant-ph].

[204] Jin Xie, Liang Gao, Kunkun Peng, Xinyu Li, and Li Haoran. "Review on flexible job shop scheduling". In: *IET Collaborative Intelligent Manufacturing* 1.3 (Aug. 2019), pp. 66–67. DOI: 10.1049/iet-cim.2018.0009.

[205] Ed Younis, Koushik Sen, Katherine Yelick, and Costin Iancu. *QFAST: Quantum Synthesis Using a Hierarchical Continuous Circuit Space*. 2020. arXiv: 2003.04462 [quant-ph].

[206] Zhongqi Zhao, Lei Fan, and Zhu Han. "Hybrid Quantum Benders' Decomposition For Mixed-Integer Linear Programming". In: *2022 IEEE Wireless Communications and Networking Conference (WCNC)*. Austin, TX, USA: IEEE Press, 2022, pp. 2536–2540. DOI: 10.1109/WCNC51071.2022.9771632.

[207] Fulai Zhu, Peiyu Xu, and Jiahao Zong. "Moore's Law: The potential, limits, and breakthroughs". In: *Applied and Computational Engineering* 10 (Sept. 2023), pp. 307–315. DOI: 10.54254/2755-2721/10/20230038.

CURRICULUM VITÆ

Anna Maria (Anneriet) KROL

17-01-1995	Born in Leiden, The Netherlands.

EDUCATION

2006–2013	High School, gymnasium cum laude Bonaventuracollege, Leiden, The Netherlands
2013–2017	Bachelor of Science in Aerospace Engineering Delft University of Technology, The Netherlands
2015–2016	Minor: Electrical Engineering for Autonomous Exploration Robots Delft University of Technology, The Netherlands
2016–2017	Minor: Electro-Mechanical System Design Erasmus exchange (6 months) at Aalborg University, Denmark
2017–2019	Master of Science in Computer Engineering Delft University of Technology, The Netherlands
2020–2024	Ph.D. in Quantum and Computer Engineering Delft University of Technology, The Netherlands
2023	Visiting researcher at BMW AG München, Germany

List of Publications

7. **A. M. Krol** and Z. Al-Ars (2024). "Beyond Quantum Shannon: Circuit Construction for General n-Qubit Gates Based on Block ZXZ-Decomposition." In: *Physical Review Applied* 22 (3 Sept. 2024), p. 034019, DOI: 10.1103/PhysRevApplied.22.034019, arXiv: 2403.13692.

6. **A. M. Krol**, M. Erdmann, E. Munro, A. Luckow and Z. Al-Ars. (2024). "Assessing the Requirements for Industry Relevant Quantum Computation." In: *Proceedings of the 2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Montréal, Canada, 2024, arXiv: 2408.02587.

5. **A. M. Krol**, M. Erdmann, R. Mishra, P. Singkanipa, E. Munro, M. Ziolkowski, A. Luckow and Z. Al-Ars. (2024) "QISS: Quantum Industrial Shift Scheduling Algorithm." Submitted to *IEEE Transactions on Quantum Engineering*, arXiv: 2401.07763.

4. **A. M. Krol**, K. Mesman, A. Sarkar, M. Möller and Z. Al-Ars. (2023). "Efficient Parameterised Compilation for Hybrid Quantum Programming." In: *Proceedings of the 2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Bellevue, WA, USA, 2023, pp. 103–111, DOI: 10.1109/QCE57702.2023.10192, arXiv: 2208.07683.

3. **A. M. Krol**, A. Sarkar, I. Ashraf, Z. Al-Ars and K. Bertels. (2022). "Efficient decomposition of unitary matrices in quantum circuit compilers." In: *Applied Sciences*, 12(2), 759, DOI: 10.3390/app12020759, arxiv: 2101.02993.

2. N. Khammassi, I. Ashraf, J. V. Someren, R. Nane, **A. M. Krol**, M. A. Rol, L. Lao, K. Bertels and C. G. Almudever. (2022). "OpenQL: A Portable Quantum Programming Framework for Quantum Accelerators." In: *Journal on Emerging Technologies in Computing Systems* 18, 1, Article 13 (January 2022), 24 pages, DOI: 10.1145/3474222, arXiv: 2005.13283.

1. K. Bertels, A. Sarkar, T. Hubregtsen, M. Serrao, A. A. Mouedenne, A. Yadav, **A. M. Krol** and I. Ashraf. (2020). "Quantum Computer Architecture: Towards Full-Stack Quantum Accelerators." In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2020, pp. 1–6, DOI: 10.23919/DATE48585.2020.9116502, arXiv: 1903.09575.

Propositions

accompanying the dissertation

PROGRAMMING QUANTUM COMPUTERS

by

Anna Maria KROL

1. Quantum utility will not be achieved without improving the performance of the classical compilers, optimizers, micro-controllers and other classical technologies required for quantum computing. This proposition pertains to this dissertation.
2. Current quantum programming languages are not suited for achieving quantum utility. This proposition pertains to this dissertation.
3. Quantum computing will only be commercially successful if Shor's algorithm is generalized to unstructured search problems. This proposition pertains to this dissertation.
4. Quantum computing will not solve the world's biggest problems.
5. All journals should have strict guidelines for what mathematical notations can be considered "common knowledge" in a field.
6. PhD candidates are among the best people to ask in-depth questions about any subject, even those not related to their research area.
7. Although many quantum researchers want standardization in quantum computing, it is still too early and will lead to stagnation.
8. Although this should not be the case, the attractiveness of graphs and figures does matter as much as the content.
9. For communication in an international context, there should be an "international" English that borrows some features from the native languages of its many speakers to improve clarity. This "international" English would include a more Dutch or German approach to compound words, for example, so that "high school" is written as "highschool" and "gate count" as "gatecount".

These propositions are regarded as opposable and defendable, and have been approved as such by the promotors prof. dr. H. P. Hofstee and dr. ir. Z. Al-Ars.

Stellingen

behorende bij het proefschrift

PROGRAMMING QUANTUM COMPUTERS

door

Anna Maria KROL

1. Quantumvoordeel zal niet worden bereikt zonder de prestaties van de klassieke compilers, optimizers, microcontrollers en andere klassieke technologieën die nodig zijn voor quantumcomputers te verbeteren. Deze stelling heeft betrekking op dit proefschrift.
2. Huidige quantumprogrammeertalen zijn niet geschikt om quantumvoordeel te bereiken. Deze stelling heeft betrekking op dit proefschrift.
3. Quantumcomputers zullen alleen commercieel succesvol zijn als Shor's algoritme wordt gegeneraliseerd naar ongestructureerde zoekproblemen. Deze stelling heeft betrekking op dit proefschrift.
4. Quantumcomputers zullen de grootste problemen ter wereld niet oplossen.
5. Alle tijdschriften zouden strikte richtlijnen moeten hebben voor welke wiskundige notatie kan worden beschouwd als "algemene kennis" in een vakgebied.
6. PhD-kandidaten behoren tot de beste mensen om diepgaande vragen te stellen over elk onderwerp, zelfs onderwerpen die niet gerelateerd zijn aan hun onderzoeksgebied.
7. Hoewel veel quantumonderzoekers standaardisatie op het gebied van quantumcomputers willen, is het nog te vroeg hiervoor en zal dit leiden tot stagnatie.
8. Hoewel dit niet het geval zou moeten zijn, is de aantrekkelijkheid van grafieken en figuren net zo belangrijk als de inhoud.
9. Voor communicatie in een internationale context zou er een "internationaal" Engels moeten zijn dat enkele kenmerken leent van de moedertalen van haar vele sprekers om de duidelijkheid te verbeteren. Dit "internationale" Engels zou bijvoorbeeld een meer Nederlandse of Duitse benadering van samengestelde woorden gebruiken, zodat "high school" wordt geschreven als "highschool" en "gate count" als "gatecount".

Deze stellingen worden opponeerbaar en verdedigbaar geacht en zijn als zodanig goedgekeurd door de promotors prof. dr. H. P. Hofstee en dr. ir. Z. Al-Ars.