RWTH AACHEN UNIVERSITY
Chair of Computer Science 2
Software Modeling and Verification

Master Thesis

Compilation of Quantum Programs with Control Flow Primitives in Superposition

Sascha Thiemann
Matr.-No.: 406187
Study Program: Computer Science M.Sc.
August 4, 2024

Supervisors: apl. Prof. Dr. Thomas Noll
Chair for Software Modeling and Verification
RWTH Aachen University

Prof. Dr. rer. nat. Dominique Unruh
Chair for Quantum Information Systems
RWTH Aachen University

Contents

1 Introduction

- Introduction with random citation to not cause error [ACR*10]

2 Background

... Background

2.1 Quantum Computing

While computers are prevalent and important in todays society, there are many relevant problem which classical computers can currently and perherps will never realistically be able to solve. Quantum Computing is gaining more momentum as the technology that could solve at least some of these problems. For example, Quantum algorithms like Shor's algorithm [Shor97] could provide a significant improvement for prime factorization given sufficient technology. Therefore, it is estimated to be a valuable market with many of the largest technology companies as well as governments investing billions in the research and development of quantum technology [RDB*22]. While there already exist detailed theoretical foundations [van20, Ying11, YYF12] and advanced algorithms for QC [ACR*10, BGB*18, LoCh19, Shor97], the technology of quantum computers is said to be on the level of classical computer in the 1950s [CFM17]. In the following section, we take a look at the basic concepts of a quantum computer and the core principles it relies on.

Classical Computers are based on simple operations, like `and`, `or`, and `not`, on bits. These bits can either have a value of 0 or 1. Similarly, at their core, quantum computers apply simple operations, like `controlled not`, and `hadamar`, on quantum bits (qubits). In contrast to classical bits, quantum computers use the unique properties of quantum mechanics to enable qubits to have not just one value of either 0 or 1 but a combination of both. Additionally, quantum computers also use the idea of entanglement to their advantage where the value of a qubit is dependent on another qubit. The combination of superposition and entanglement enable quantum computers to solve specific problems more efficiently than classical computers [RDB*22], e.g. prime factorization [Shor97].

citation needed?

Repeating info from paragraph above?

Different QC models: analog/digital -> focus on digital, use [DiCh20]

2.1.1 Superposition

The concept of superposition is most known for its role in the "Schrödinger's cat" thought experiment [**?**] where the life of a cat is dependent on a particle in superposition. Similar to the cat being refered to as alive and dead at the same time, qubits in superposition are often informally described as simultaniously having a value of 0 and 1. However, a qubit in superposition is more formally a linear combination of its basis states. The basis states are the states where the qubit has a value of 0, written $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, and 1, written $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. Therefore, a state ψ in superposition can be written as:

also describe the definition of $|+\rangle, |-\rangle$?

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

The factors α and β are the amplitudes of the basis states and are complex numbers. The probability that the state collapses to a basis state when measured is related to be corresponding amplitude of that basis state and can be computed by squaring it. Because a state will always collapse to a basis state when measured, the sum of the squares of the amplitudes must always be equal to one for the state to be a valid quantum state.

formulate at math formula to be easier to read?

2.1.2 Entanglement

- Explain entanglement
- How is entanglement relevant for quantum computing?
- Explain disruptive entanglement

2.1.3 Operators and Gates

- General theoretical bases of operators and gates
- Most important gates and their functions

2.1.4 Measurement

- How are qubits measured?
- What is the effect of measurement on qubits?

2.1.5 Relevant Algorithms

- Shortly describe algorithms referenced later
- QFT (Quantum Fourier Transform)
- Shor's algorithm

2.2 Quantum Control Flow

- Introduction into quantum control flow
- Branching
- Iteration
- Limitations

2.2.1 Branching

- Explain branching principle
- How is branching relevant for control flow?
  - Example of branching in classical computing
  - Example of branching in quantum computing

2.2.2 Iteration

2.2.3 Limitations

Reversibility

- Explain reversibility principle
- How is reversibility relevant for control flow?
  - Example of reversible and irreversible operations (JMP instructions)
-

Synchronization

- Explain synchronization principle
- Tortoise and hare example

2.2.4 Quantum Control Machine

- Definition of Quantum Control Machine
- How does it solve/handle the limitations of quantum control flow?
- Example program
  - Example for non-reversible program
  - Example for reversible but not synchronous program
  - Example for correct program

```
1       add   res $1
2       add   r1  y
3  l1:  rjne  l3  r1  y
4  l2:  jz    l4  r1
5       mul   res x
6       radd  r1  $1
7  l3:  jmp   l1
8  l4:  rjmp  l2
```

Figure 2.1: QCM exponentiation without synchronization

```
1       add   res $1
2       add   r1  max
3  l1:  rjne  l3  r1  max
4  l2:  jz    l4  r1
5  l5:  jg    l7  r1  y
6       mul   res x
7  l6:  jmp   l8
8  l7:  rjmp  l5
9       nop             ; padding
10 l8:  rjle  l6  r1  y
11      radd  r1  $1
12 l3:  jmp   l1
13 l4:  rjmp  l2
```

Figure 2.2: Synchronized QCM exponentiation

*

2.3 Quantum Languages

2.3.1 QASM Language

- Give overview of QASM language and concepts

2.4 Compilation

2.4.1 Lexer

2.4.2 Parser

2.4.3 Semantic Analysis

2.4.4 Code Generation

2.4.5 Optimization

- Different optimization techniques
  - Constant folding or constant propagation
  - Peephole optimization

2.4.6 ANTLR

- Give overview of ANTLR and parsing in general

3 Concept

- Expand on concept section from proposal:

The concept for the master thesis is to take the idea of the QCM, specifically the core concept of quantum control flow, and reduce it to its most basic elements and make it realistic for and applicable to NISQ era quantum computers. Concretely, we want to go from jump instructions to basic if-else clause to reduce the complexity of the code and make it easier to read and write. These if-else clauses can easily be implemented as the application of controlled gates. Moreover, because of the synchronization principle and the fact that current quantum computer technology does not support loops depending on measurement, any other loop can be reduced to a for-loop that is unrolled at compile time.

To achieve this goal, we want to define a language "Luie" (short for loop-unrolled if-else) which is partially based on the quantum while language used by Ying [Ying11]. The language is extended by a quantum if clause which takes a quantum register and executes the statements in the clause based on the value of the register. Furthermore, the clause could even be extended to include the evaluation of boolean expression. While the language cannot include while statements based on measurements of registers, as it is the case in the language proposed by Ying, it can include bounded loops which are unrolled at compile time. The language will then be compiled to QASM.

4 Implementation

4.1 Semantic analysis

- What is semantic analysis used for?
- How is it implemented in Luie?
- Different types of semantic analysis
- Errors
  - Types of errors: Critical, warning
  - Different critical errors (Type, undefined, ...)
  - Different warnings (invalid range, ...)

4.2 Code Generation

- How is code generated?

- Important classes and abstractions

4.3 Language Overview

4.3.1 Blocks and Scopes

- Basic structure of Luie
- Consists of blocks and statements
- One main block
- Symbol table that handles scopes
- All blocks have scope

Grammar

- A block consists of arbitrarily many definitions and statements

```
1     grammar Luie;
2
3     parse
4      : block EOF
5      ;
6
7     block
8      : (definition | statement)*
9      ;
```

4.3.2 Data Types

- Different data types
  - Register
  - Qubits (Registers with size 1)
  - Iterators, in more detail in Sec. 4.3.4

Grammar

- Registers and qubits defined in declaration
- Optional size, otherwise qubit (size 1).
- Identifier is the name of the register/qubit
- Arbitrary string starting with a character or underscore

- Later

```
1     declaration
2      : 'qubit' ('[' size=expression ']')?
           IDENTIFIER ';'
3      ;
4
5     IDENTIFIER
6       : [a-zA-Z_] [a-zA-Z_0-9]*
7       ;
```

Semantic analysis

- Semantic analysis can be differentiated between checking the use of an identifier and checking the type of an identifier.

- *Definedness*: symbol table to check if an identifier is defined

- *Type checking*: symbol table to check if the type of an identifier is correct
  - What are different type
  - Object oriented approach, i.e. because qubit inherits from register it is also a register
  - -> less checks required

Code Generation

- `Definition`s used for code generation, symbols can be transformed into definitions

- Needs unique identifier (language has scopes -> multiple variables with same name possible, while QASM does not)

- Unique identifiers given at generation time because of for loops
  - Explain why ...

4.3.3 Gate Application

Grammar

Semantic Analysis

Code Generation

4.3.4 Control Flow

Grammar

Code Generation

4.3.5 Expressions

Grammar

- Consists of expressions, terms and factors
  - Expressions consist of expression, operator, and term or just a term
  - Term consists of term, operator, and factor or just a factor
  - Factor consists of expression in parentheses, a negated factor, number, identifier or function call
- Inherent order of operations

Evaluation

- All expressions evaluated at compile time
- All expressions inherit from abstract `Expression` class, which has an `Evaluate` method
- Generic return type `T`
- ...

4.4 Optimization

4.4.1 Constant folding

- Inherent in the language
- All variables known at compile time
- Any expression is evaluated at compile time

4.4.2 Peephole optimization

- Not implemented, but planed
- Replace sequences of gates with more efficient ones

4.5 Test Cases

- Different test categories
- How are they implemented?
- What do they test?
- (Continuous integration)

5 Conclusion and Future Work

- Conclusion to thesis
- Future work
  - how could language be extended

Bibliography

[ACR*10] A. Ambainis, A. M. Childs, B. W. Reichardt, R. Špalek, and S. Zhang. Any and-or formula of size n can be evaluated in time $n^1/2+o(1)$ on a quantum computer. *SIAM Journal on Computing*, 39(6):2513–2530, 2010.

[BGB*18] Ryan Babbush, Craig Gidney, Dominic W. Berry, Nathan Wiebe, Jarrod McClean, Alexandru Paler, Austin Fowler, and Hartmut Neven. Encoding electronic spectra in quantum circuits with linear t complexity. *Physical Review X*, 8(4), 2018.

[CFM17] Frederic T. Chong, Diana Franklin, and Margaret Martonosi. Programming languages and compiler design for realistic quantum hardware. *Nature*, 549(7671):180–187, 2017.

[DiCh20] Yongshan Ding and Frederic T. Chong. *Quantum Computer Systems*. Springer International Publishing, Cham, 2020.

[LoCh19] Guang Hao Low and Isaac L. Chuang. Hamiltonian simulation by qubitization. *Quantum*, 3:163, 2019.

[RDB*22] Roman Rietsche, Christian Dremel, Samuel Bosch, Léa Steinacker, Miriam Meckel, and Jan-Marco Leimeister. Quantum computing. *Electronic Markets*, 32(4):2525–2536, 2022.

[Shor97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[van20] John van de Wetering. Zx-calculus for the working quantum computer scientist.

[Ying11] Mingsheng Ying. Floyd–hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems*, 33(6):1–49, 2011.

[YYF12] Mingsheng Ying, Nengkun Yu, and Yuan Feng. Defining quantum control flow.