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

Master Thesis Proposal

Title tbd

Sascha Thiemann
Matr.-No.: 406187
Study Program: Computer Science M.Sc.
June 25, 2024

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

Contents

1 Introduction

Quantum algorithms like Shor's algorithm [Shor97] could provide a significant improvement to classical solutions given sufficient technology. Therefore, a lot of research is conducted in the area of Quantum Computing (QC). 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]. Currently, there are many architectures and ideas for the future of QC. In this proposal and the following thesis, we want to build upon the Quantum Control Machine proposed by Yuan et al. [YVC24] and discuss the idea of quantum control flow in general.

Quantum control flow can be divided into *quantum branching* and *iteration*. [YVC24] First defined by Ying et al. [YYF12], quantum branching is based on Dijkstra's guarded clauses [Dijk75]. Guarded clauses concern the nondeterministic executing of functions based on boolean expressions. In contrast, quantum branching allows for execution of functions based on a value in superposition such that the result may be a superposition of the results of the individual functions. Quantum branching is, e.g., used in simulation algorithms like [BGB*18], and [LoCh19]. Extending on the idea of branching, quantum iterations is the repetition of a function based on a value in superposition.

2 Motivation

With the emergence of quantum computing, many quantum languages were introduced. Most languages focus on a lower level representation of quantum circuits. An example is the popular Open Quantum Assembly Language (QASM)[CBSG17]. QASM consists mainly of quantum and classical gates that can be manipulated by predefined and composite gates as well as limited classical if-statements. In contrast, there are also languages with a focus on high level interactions, e.g. Tower[ChMi22] which contains data structures in superposition, and Silq [BBGV20] which allows for automatic uncomputing of registers. What all these languages have in common is the restriction to quantum data while using only classical control flow. Although quantum control flow was defined by Ying et al. [YYF12] over ten years ago, only very few languages have incorporated the principle. One example is the functional programming language proposed by Altenkirch et al. [AlGr05] where quantum branching is used to define, e.g., the Hadamard gate, as proposed by Ying et al. [YYF12]. Only recently was the Quantum Control Machine (QCM) with quantum control flow at its core proposed by Yuan et al. [YVC24].

The QCMs syntax and logic are both heavily influenced by classical assembly languages. The language consists of quantum registers, gate, swap and get-bit operations[1], simple numeric operations on registers, and, finally, jump instructions. The jump instructions range from simple to conditional to indirect and are used to enable quantum control flow. Although the jump instructions are based on jumps in classical computers, they are limited by two concepts quantum computers based on unitary gates must adhere to, *reversibility* and *synchronization*. [YVC24]

When quantum computers are based on unitary gates, all their operations need to be unitary and, therefore, reversible as well. This also includes jump instructions which are not reversible in classical computers. To ensure reversibility of jumps, the QCM uses a *branch control register* whose value controls how much the instruction pointer of the machine advances after each instruction. The branch control register can then be manipulated reversibly by, e.g., adding or subtracting from it. The idea of a branch control register can also be found in reversible architectures for classical machines [AGY07, TAG12].

Although such a program counter addresses the issue of reversibility, it can become entangled with data registers when in superposition. This can lead to disruptive entanglement where the output of the program becomes invalid [YVC24]. To prevent any disruptive entanglement of the data and control registers, the QCM adheres to the principle of synchronization. It requires that the control flow is separated from the data at the end of execution. Examples where synchronization comes into play are given in Fig. 2.1 and Fig. 2.2 where x^y and $x^{\texttt{min}\{y,max\}}$ are calculated respectively. While the first example is completely reversible, it does not adhere to the principle of synchronization. Given two different inputs, the loop will be executed a different number of times. This means that after the faster of the two programs completed the loop, the program counter of the slower one cannot catch up. To prevent this issue, the second program uses padding which is executed instead of the main loop.

Because of the reversibility of the QCM, any jump instruction in the code needs to have an opposing return jump instruction. Additionally, the synchronization principle requires any loop with n instructions to contain n padding instructions and any loops cannot depend on quantum data for their iterations. Together with the syntax based on classical assembly languages, the language of the QCM is hard to read and write.

[1]The gate operations are limited to the Hadamard and NOT gates.

```
     add   res $1
     add   r1  y
l1:  rjne  l3  r1  y
l2:  jz    l4  r1
     mul   res x
     radd  r1  $1
l3:  jmp   l1
l4:  rjmp  l2
```

Figure 2.1: QCM exponentiation without synchronization

```
     add   res $1
     add   r1  max
l1:  rjne  l3  r1  max
l2:  jz    l4  r1
l5:  jg    l7  r1  y
     mul   res x
l6:  jmp   l8
l7:  rjmp  l5
     nop               ; padding
l8:  rjle  l6  r1  y
     radd  r1  $1
l3:  jmp   l1
l4:  rjmp  l2
```

Figure 2.2: Synchronized QCM exponentiation

3 Concept

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. A basic grammar for the language can be seen in Appendix .1.

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.

[AlGr05] T. Altenkirch and J. Grattage. A functional quantum programming language. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*, pages 249–258. IEEE, 2005.

[AGY07] Holger Bock Axelsen, Robert Glück, and Tetsuo Yokoyama. Reversible machine code and its abstract processor architecture. In Volker Diekert, Mikhail V. Volkov, and Andrei Voronkov, editors, *Computer Science – Theory and Applications*, volume 4649 of *Lecture Notes in Computer Science*, pages 56–69. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[BBGV20] Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin Vechev. Silq: a high-level quantum language with safe uncomputation and intuitive semantics. In Alastair F. Donaldson and Emina Torlak, editors, *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 286–300, New York, NY, USA, 2020. ACM.

[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.

[CBSG17] Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. Open quantum assembly language.

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

[ChMi22] Charles Yuan and Michael Carbin. Tower: Data structures in quantum superposition, 2022.

[Dijk75] Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.

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

[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.

[TAG12] Michael Kirkedal Thomsen, Holger Bock Axelsen, and Robert Glück. A reversible processor architecture and its reversible logic design. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Alexis de Vos, and Robert Wille, editors, *Reversible Computation*, volume 7165 of *Lecture Notes in Computer Science*, pages 30–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[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.

[YVC24] Charles Yuan, Agnes Villanyi, and Michael Carbin. Quantum control machine: The limits of control flow in quantum programming. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA1):1–28, 2024.

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

.1 Grammar

```
grammar Luie;

parse
 : block EOF
 ;

block
 : (definition | statement)*
 ;

definition
 : 'qubit' IDENTIFIER ';'
 ;

statement
 : GATE IDENTIFIER ';'
 | qifStatement
 ;

qifStatement
 : ifStat elseStat? END
 ;

ifStat
 : IF IDENTIFIER DO block
 ;

elseStat
 : ELSE DO block
 ;

GATE
 : XGATE
 | ZGATE
 | HGATE
 ;

XGATE : 'x';
ZGATE : 'z';
HGATE : 'h';

IF      : 'qif';
ELSE    : 'else';
DO      : 'do';
END      : 'end';

IDENTIFIER
 : [a-zA-Z_] [a-zA-Z_0-9]*
 ;

COMMENT
```

```
      : ( '//' ~[\r\n]* | '/*' .*? '*/' ) ->
          skip
      ;

    SPACE
      : [ \t\r\n\u000C] -> skip
      ;
```