Compilation of Quantum Programs with Control Flow Primitives in Superposition

Master Thesis

05.02.25

Sascha Thiemann

Supervisors: apl. Prof. Dr. Thomas Noll

Prof. Dr. rer. nat. Dominique Unruh

Outline

Introduction

test [Aaby, 2003]

Quantum Control Flow

- The idea of Quantum Control Flow was first used by [Altenkirch and Grattage, 2005] to define function quantum programming language.
- For example, it was used to define the Hadamard gate as the function *had*:

$$
\begin{aligned}
\mathit{had} : Q &\to Q \\
\mathit{had} : x &\mapsto \ \texttt{if}^{\circ} x \\
&\qquad \texttt{then} \ \{\mathit{false} \mid -\mathit{true}\} \\
&\qquad \texttt{else} \ \{\mathit{false} \mid \mathit{true}\}
\end{aligned}
$$

- Later, the concept was formally defined by [Ying et al., 2012].
- Quantum branching allows fot the execution of function based on values in superposition.
- The result is the superposition of the results of individual executions.

Limitations — Reversibity

- Quantum control flow is mainly limited by two principles: *reversibility* and *synchronization*.
- Any sequence of instructions on gate-based quantum computers, excluding measurements, is required to be reversible by definition, as they are all unitary transformations.
- As a result, control flow, as implemented in classical computers, is not possible.
- For example, any classical jump instruction is inherently irreversible.
- Landauer Embedding [Landauer, 1961] seems to offer solution.
- The embedding can turn any non-reversible function into a reversible one by not only returning the output but also the input of the function.
- For example, any non-reversible function $f : D \to D'$ can be given as a reversible function $g : D \to D' \times D$ with $g(x) = (f(x), x)$.
- However, because the output is the result together with the program history and the result depends on the history, they become entangled.
- This leads to disruptive entanglement [Yuan et al., 2024].

Background

Limitations — Synchronization

- The program counter can become entangled with the data and result in disruptive entanglement leading to an invalid result.
- The principle of synchronization states that control flow must become independent from the data.
- For example, loops cannot depend solely on value in superposition.
- *Tortoise and hare* problem
- Instead, a loop must be bounded by a classical value [Yuan et al., 2024].

Quantum Control Machine

- Quantum Control Machine (QCM), proposed by [Yuan et al., 2024], is an instruction set architecture, focused on quantum control flow.
- Both its syntax and logic are similar to classical assembly language, utilizing (conditional) jump instructions.
- The architecture employs a branch control register *bcr* to enable reversible jump instructions.

Intructions

Operation	Syntax	Semantics[1]
No-op	`nop`	Only increases instruction pointer by the *bcr*.
Addition	`add` *ra rb*	Adds register *rb* to *ra*.
Multiplication	`mul` *ra rb*	Multiplies register *ra* by *rb*.
Jump	`jmp` *p*	Increases *bcr* by *p*.
Conditional Jumps	`jz` *p ra*	Increases *bcr* by *p* if *ra* is 0.
	`jne` *p ra rb*	Increases *bcr* by *p* if *ra* is not equal to *rb*.

[1] After all operations, the instruction pointer is increased by the value of the *bcr*.

An excerpt of the QCM instruction set with instructions used in later examples.

Language Overview

. . .

Language

Syntax

...

Translation

. . .

Implementation

Code Generation

...

Optimization

...

Evaluation

. . .

Conclusion

References

Aaby, A. A. (2003).
Compiler construction using flex and bison.

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

Landauer, R. (1961).
Irreversibility and heat generation in the computing process.
IBM Journal of Research and Development, 5(3):183–191.

Ying, M., Yu, N., and Feng, Y. (2012).
Defining quantum control flow.

Yuan, C., Villanyi, A., and Carbin, M. (2024).
Quantum control machine: The limits of control flow in quantum programming.

References

Proceedings of the ACM on Programming Languages, 8(OOPSLA1):1–28.