ATPL 2024: Hybrid Quantum-Classical Programming.

Lecture 2: Quantum bits – calculi and languages

Michael Kirkedal Thomsen (shapper@diku.dk)

November 19, 2024

Outline

Table of Contents

Recap: The need for Hybrid Programming Models, Languages and Compilers

Thus, we will need:

- Hybrid **programming languages** that allow us to express the QC and CC parts and their interactions.
- Hybrid **programming models** and formal semantics that allow us to reason about hybrid programs.
- Hybrid **compilers** that can optimize the quantum and classical parts of the computation, and ultimately produce code that runs on a combination of classical HPC and quantum accelerator hardware, and orchestrates their interactions.

This does *not* currently exist. You will take part in building this brave new world!

Computation Models and Programming Languages

- What do you think when we say computation model?
- What computation models do you know?

Computation Models and Programming Languages

- What do you think when we say computation model?
- What computation models do you know?
- What is the difference between model and a language?

Models and languages

Model of computation

A computation model is a (theoretical construct to) description of how an output of a mathematical function is computed given an input.
Furthermore, it can describes how units of computations, memories, and communications are organized.

Examples: Turing machines, automata, Von Neumann machine, random-access machine

Programming Language

Programming languages are used as a notion of defining computations
They are described in terms of their syntax (form) and semantics (meaning), usually defined by a formal language.
Languages often provide features such as a type system, variables, and mechanisms for error handling.

How does the real world fit into this?

How does the real world fit into this?

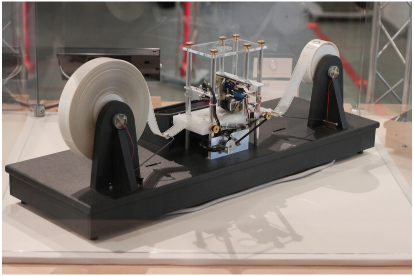
- We want implementable
  - It is called a computation model

Why is the Turing "successful"?

How does the real world fit into this?

- We want implementable
  - It is called a computation model

Why is the Turing "successful"?



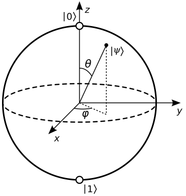
- It does reflect the reality and generalise.
- We can implement other languages on it and vice versa.
- Idealised complexity; does not give the true world.
- Has a notion of infinity.

Table of Contents

What do we need in a quantum computing model?

What do we need in a quantum computing model?

Considering the “classical” quantum circuit model:

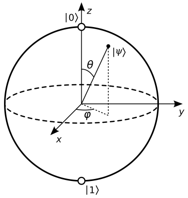


The qubit

- Modeling a quantum state, unit vector of the Bloch sphere.
- Relate to wave functions using position or momentum variables

What do we need in a quantum computing model?

Considering the "classical" quantum circuit model:



The qubit

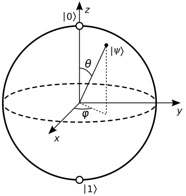
- Modeling a quantum state, unit vector of the Bloch sphere.
- Relate to wave functions using position or momentum variables

Operators (gates)

- Describe the evolution of quantum states
- Require: unitary, information preserving.

What do we need in a quantum computing model?

Considering the “classical” quantum circuit model:



The qubit

- Modeling a quantum state, unit vector of the Bloch sphere.
- Relate to wave functions using position or momentum variables

Operators (gates)

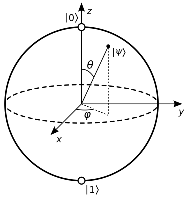
- Describe the evolution of quantum states
- Require: unitary, information preserving.

Composition

- Parallel, sequential
- Require: no-cloning

What do we need in a quantum computing model?

Considering the "classical" quantum circuit model:



The qubit

- Modeling a quantum state, unit vector of the Bloch sphere.
- Relate to wave functions using position or momentum variables

Operators (gates)

- Describe the evolution of quantum states
- Require: unitary, information preserving.

Composition

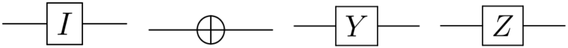
- Parallel, sequential
- Require: no-cloning

Quantum Circuits I

Qubits

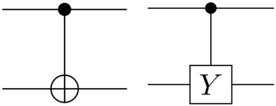
- We will have that as lines

Quantum gates:



Some quantum gates: Identity gate, NOT gate, Pauli Y, and Pauli Z.

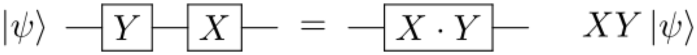
Controlled-gates



Some circuit diagrams of controlled Pauli gates: CNOT and controlled-Y.

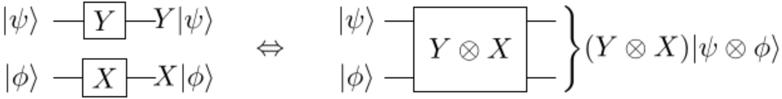
Quantum Circuits II

Sequential composition:



Two gates Y and X in series

Parallel composition:



Two gates Y and X in parallel

Linear algebra and bra-ket (Dirac) notation I

The squared amplitudes in a quantum state are the probabilities of measuring each basis state, so given a qubit state before and after acting with a program P:

$$|\Psi\rangle = \sum_{i=0}^{2^n-1} a_i |i\rangle \quad \text{and} \quad P|\Psi\rangle = \sum_{i=0}^{2^n-1} b_i |i\rangle$$

we must have $1 = a_0^2 + \ldots + a_{2^n-1}^2 = b_0^2 + \ldots + b_{2^n-1}^2$.

$$\begin{array}{c c} & \begin{array}{cccc} 00 & 01 & 10 & 11 \end{array} \\ \begin{array}{c} 00 \\ 01 \\ 10 \\ 11 \end{array} & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$|10\rangle = |11\rangle (a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle) = a_{00}|00\rangle + a_{01}|01\rangle + a_{11}|10\rangle + a_{10}|11\rangle$$

Linear algebra and bra-ket (Dirac) notation II

Sequential composition:

- Matrix multiplication

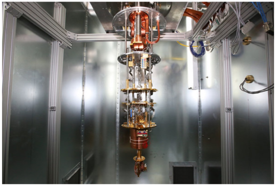
Parallel composition:

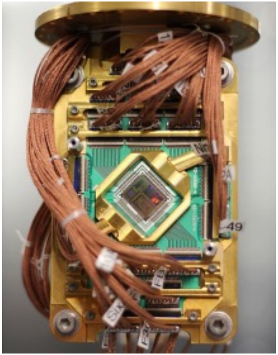
$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \otimes \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} = \begin{bmatrix} a_{1,1} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{1,2} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \\ a_{2,1} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{2,2} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} & a_{1,2}b_{1,1} & a_{1,2}b_{1,2} \\ a_{1,1}b_{2,1} & a_{1,1}b_{2,2} & a_{1,2}b_{2,1} & a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} & a_{2,1}b_{1,2} & a_{2,2}b_{1,1} & a_{2,2}b_{1,2} \\ a_{2,1}b_{2,1} & a_{2,1}b_{2,2} & a_{2,2}b_{2,1} & a_{2,2}b_{2,2} \end{bmatrix}.$$

Table of Contents

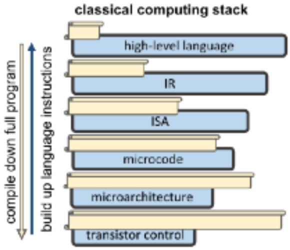
Quantum Computers

- Superconducting
- Trapped ions
- Photonic
- Neutral atom
- Silicon spin

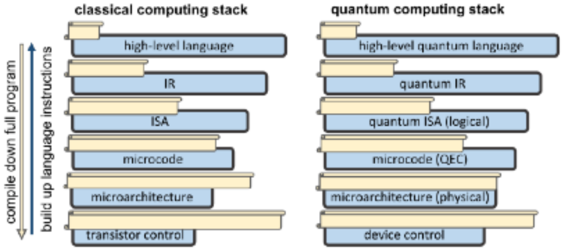




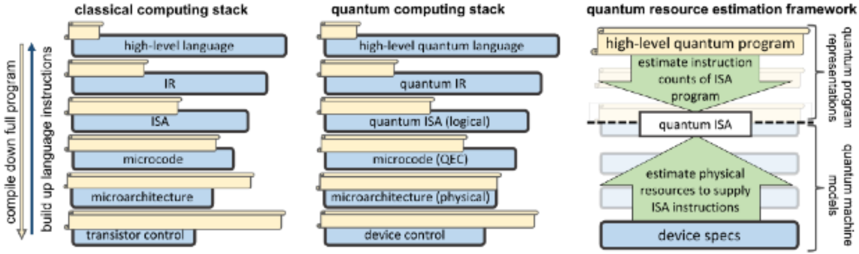
Quantum Stack



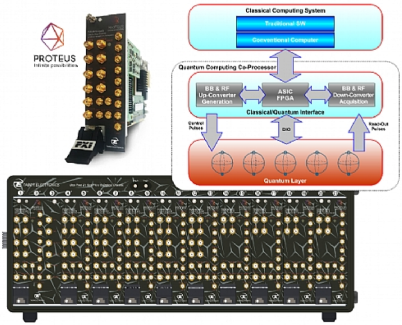
Quantum Stack



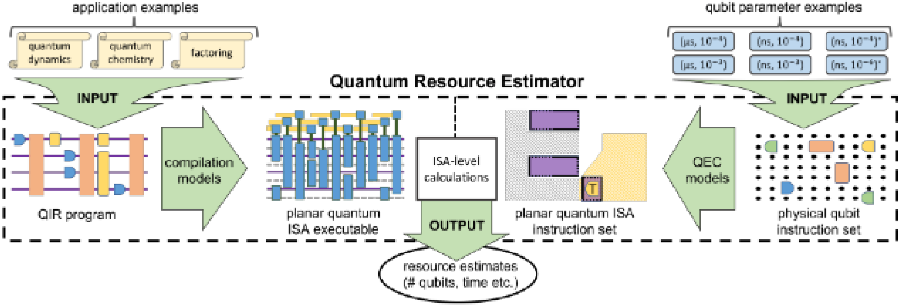
Quantum Stack



M. Beverland et. al, arXiv:2211.07629


PROTEUS
Infinite possibilities
PXI
Classical Computing System
Traditional SW
Conventional Computer
Quantum Computing Co-Processor
BB & RF
Up-Converter
Generation
ASIC
FPGA
BB & RF
Down-Converter
Acquisition
Classical/Quantum Interface
Control
Pulses
DIO
Read-Out
Pulses
Quantum Layer

Need for compilation already without hybrid



This picture is mainly to give an example. There are many approaches and no one the "right" one yet.

Compilation steps

- Generation of intermediate representation
  - Generate QASM code
- Logic Synthesis
  - Make simple gates from more advanced operations
- Optimise
  - E.g. reduce T-count of circuits
- Technology mapping
  - Adjust to the specific architecture
  - Possibly generate Surface codes

Table of Contents

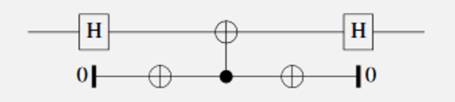
Quipper – A Haskell Library

- Published in 2013.
- An embeded language based in Haskell
  - Developed as part of the IARPA 's QCS project
- The quantum programs are written in Haskell adding the appropriate libraries
- Quipper is a circuit description language

```
import Quipper
spos :: Bool -> Circ Qubit
spos b = do
  q <- qinit b
  r <- hadamard q
  return r
```

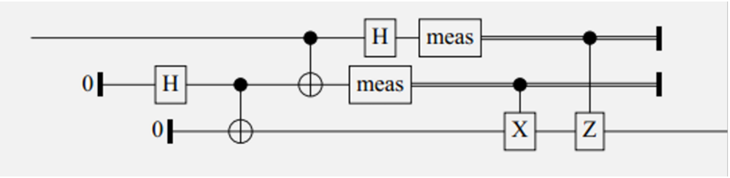
Quipper – example I

```
circ :: Qubit -> Circ Qubit
circ x = do
  hadamard_at x
  with_ancilla $ \y -> do
    qnot_at y
    qnot x `controlled` y
    qnot_at y
  hadamard_at x
  return x
```



Quipper – example II

```
teleport :: Qubit -> Circ Qubit
teleport q = do
  (a,b) <- bell00
  (x,y) <- alice q a
  b <- bob b (x,y)
  return b
```



More circuit description level languages

- QCEngine
- Qiskit
- OpenQASM
- Q#
- Cirq
- t|ket
- QuTiP, or Quantum Toolbox in Python

Examples: `https://oreilly-qc.github.io/`
`https://github.com/CQCL/pytket-docs`

Q# – Microsoft

Run in Python host program or Jupyter Notebook

```
namespace Superposition {
    @EntryPoint()
    operation MeasureOneQubit() : Result {
        // Allocate a qubit. By default, it's in the 0 state.
        use q = Qubit();
        // Apply the Hadamard operation, H, to the state.
        // It now has a 50% chance of being measured as 0 or
    1.
        H(q);
        // Measure the qubit in the Z-basis.
        let result = M(q);
        // Reset the qubit before releasing it.
        Reset(q);
        // Return the result of the measurement.
        return result;
    }
}
```

https:

Table of Contents

"Higher-level" languages and tools

- PyQuil
- Silq
- TensorFlow Quantum
- PennyLane – ML?

See more: `https://quantumzeitgeist.com/top-quantum-computing-programming-languages/`

PyQuil – from Rigetti Computing

```
import numpy as np
from pyquil import Program
from pyquil.quil import DefGate

# First we define the new gate from a matrix
sqrt_x = np.array([[ 0.5+0.5j,  0.5-0.5j],
                   [ 0.5-0.5j,  0.5+0.5j]])

# Get the Quil definition for the new gate
sqrt_x_definition = DefGate("SQRT-X", sqrt_x)
# Get the gate constructor
SQRT_X = sqrt_x_definition.get_constructor()

# Then we can use the new gate
p = Program()
p += sqrt_x_definition
p += SQRT_X(0)
print(p)
```

https://pyquil-docs.rigetti.com/en/stable/

Silq – ETH

Strongly statically typed

`https://silq.ethz.ch/`

TensorFlow Quantum

TensorFlow Quantum is a library for hybrid quantum-classical machine learning.

`https://www.tensorflow.org/quantum`

Table of Contents

Very low-level languages and tools

- QUA: a pulse-level quantum programming language
- OpenFermion: Python-based framework designed as an electronic structure package for quantum computers

See more: `https://quantumzeitgeist.com/top-quantum-computing-programming-languages/`

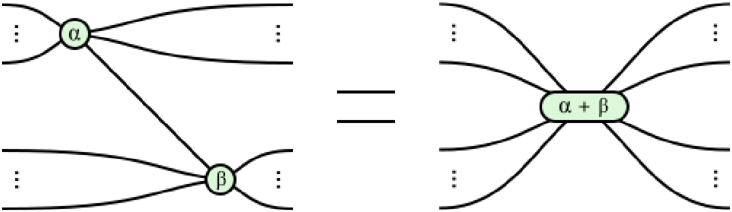
Table of Contents

Quantum models and algebra

- Quantum Turing machines
  - Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer; David Deutsch, 1985
- ZX calculus
  - Coecke, Duncan
  - `https://zxcalculus.com/`
- Programming the quantum future; Valiron, Ross, Selinger, Alexander, Smith
  - https://dl.acm.org/doi/10.1145/2699415
- Reversible quantum combinators: $\cup\Pi$; Heunen, Kaarsgaard
  - https://arxiv.org/abs/2107.12144

The ZX-calculus

- Graphical language that goes beyond circuit diagrams
- Semantic Defined over categorical theory
- `https://zxcalculus.com/`



The green spider rule, saying you can merge two green spiders if they are joined by a wire

Table of Contents

The need for Hybrid Quantum-Classical Programming

To leverage quantum computing in practice, we need to be able to:

1. Identify the sub-problems that can benefit from quantum computing.
2. Decompose the problem into quantum and classical parts and identify the interfaces and interaction.
3. Efficiently encode the input and output of the quantum sub-problems.
4. Efficiently combine the quantum and classical parts.

Quantum programming languages today

- Not near to implement a hybrid language
- Does not have the general constructs we know

We don't know how to do this — do you?