Implementing Quantum Algorithms in SML

Caroline Kierkegaard
and Mikkel Willén

Supervisor: Martin Elsman

KØBENHAVNS UNIVERSITET

Introduction

- Motivation
- Design and implementation
- Results
- Evaluation
- Future work
- Conclusion

Motivation

- Implementing quantum algorithms in classical computation framework
- Concretely, implementing and simulating Grover's Algorithm by extending the SML framework
- Motivation for choosing Grover's Algorithm
  - In an unstructured database containing N elements, find one specific element
  - Classical computer will need $O(N)$ steps
  - Quantum computer will only need $O(\sqrt{N})$ steps

Design and implementation

- Extending SML framework with building blocks used for Grover's Algorithm
  - Oracle operation
  - Diffusion operation
  - Repetition of parts of circuits
  - Working with ancillary bits

The Oracle Operation

Algorithm 1: Oracle Function

Input : A target state, n qubits

Flip target bits to map the target to $|1...1\rangle$

Combine I for 0's and X for 1's with *oo*

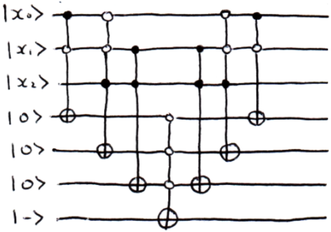
Apply a controlled not on the ancilla, where all the other bits are controls

Reverse the bit flip

- 3 ways of providing the oracle
    - Gate with target
    - Function that creates circuit
    - Circuit

Future work: Implementing an Oracle for the SAT Problem

- Simple implementation - 1 qubit per literal, 1 ancilla per clause
- Flip ancilla bit, if clause is unsatisfied
- Flip checker, if all clauses are satisfied
- Example: $(\neg x_0 \vee x_1) \wedge (x_0 \vee x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$ [1]



[1] https://cnot.io/quantum$_a$lgorithms/grover/using$_g$rovers$_a$lgorithm.html

Future work: Translate higher-level oracle functions to circuits

- Oracle functions written in higher-level languages
- Compile to quantum circuit

The Diffusion Operation

Algorithm 2: Diffusion Operator

Input : n qubits, an ancilla qubit

Apply Hadamard gates to all qubits to create a superposition

Flip all qubits using X gates

Apply a controlled X gate on the ancilla, controlled by all other qubits

Reverse the flips with X gates

Reverse the initial superposition by reapplying Hadamard gates

Implementing Grovers

- Initialise all qubits to $|0\rangle$ + ancillary bit
- Repeat Grover's iterations optimal number of times times

```
val iterations = Real.floor (Math.pi / 4.0 * Math.sqrt (Real.fromInt (powInt 2 n)))
```

```
fun repeatCircuit t n =
    if n < 1 then
        raise Fail "Number of iterations must be greater than 0"
    else if n = 1 then
        t
    else
        t oo repeatCircuit t (n - 1)
```

Implementing Grovers

- Combine all building blocks

```
fun groversNaive target n : t * ket =
    let
        val numQubits = n + 1
        val iterations = Real.floor (Math.pi / 4.0 * Math.sqrt (Real.fromInt (powInt 2 n)))
        val hadamardGates = hadamard numQubits
        val initAncilla = zAncilla numQubits
        val repetition = repeatCircuit (oracleNaive target numQubits oo diffusionNaive target numQubits) iterations
    in
        (hadamardGates oo initAncilla oo repetition, initKets n 1)
    end
```

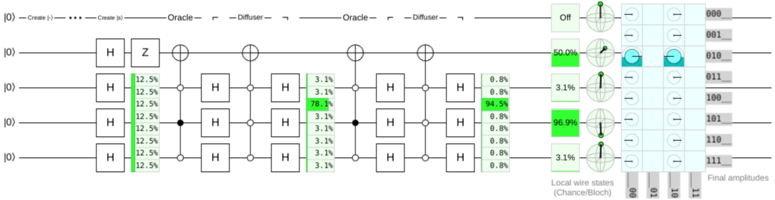
Measuring the distribution with ancillary bits

```
fun measure_dist_ancilla_helper (d: real vector) (num_ancilla: int) : real vector =
    if num_ancilla < 0 then
        raise Fail "You cannot have a negative number of ancilla bits"
    else if num_ancilla = 0 then
        d
    else
        let
            val len = Vector.length d
            val halflen = len div 2
            val v2 = Vector.tabulate (halflen, fn i => Vector.sub(d, 2 * i) + Vector.sub(d, 2 * i + 1))
        in
            measure_dist_ancilla_helper v2 (num_ancilla - 1)
        end

fun measure_dist_ancilla (s: state) (num_ancilla: int) : dist =
    let
        val v = dist s
        val v2 = measure_dist_ancilla_helper v num_ancilla
        val len = log2 (Vector.length v2)
    in
        Vector.mapi (fn (i, p) => (toKet(len, i), p)) v2
    end
```

Future work: Extending Support for Multiple Ancilla Bits

- Generality
- Use for other algorithms
- `Measure_dist_ancilla` supports multiple
- Other functions supports one
- Printing the ancilla bits



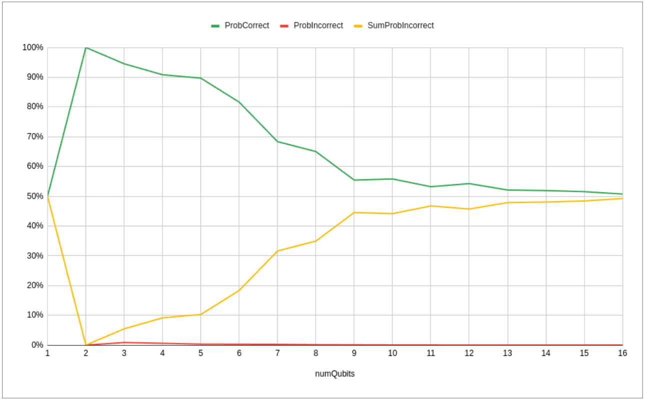
Probability distribution

- Experiments on circuit with 1-16 qubits
- Gradual decline in correct probability

numQubits	ProbCorrect	ProbWrong	SumProbWrong
1	0.5	0.5	0.5
2	1	0	0
3	0.9453125	0.0078125	0.0546875
4	0.908447265625	0.006103515625	0.0915527344
5	0.896936535835	0.00332462787628	0.1030634642
6	0.816377019397	0.00291465048576	0.1836229806
7	0.683735462787	0.00249027194656	0.31626453721
8	0.650349994728	0.00137117649126	0.34965000527
9	0.554456476626	0.000871905133804	0.44554352337
10	0.558355923306	0.000431714639975	0.44164407669
11	0.532238224051	0.000228510882242	0.46776177594
12	0.542708431802	0.000111670712625	0.45729156819
13	0.521155601617	5.84598215581E-05	0.47884439838
14	0.519292732029	2.93418340945E-05	0.48070726797
15	0.515622768257	1.47824711369E-05	0.48437723174
16	0.507572313746	7.51396484708E-06	0.49242768625

Graph of probabilities

- Individuel probabilities of wrong result declines
- but aggregate increases



Future work: Grovers Algorithm with zero theoretical failure rate

- Standard Grovers probability
- Phase inversion
- Two phase rotations through angle ϕ
- Obtain marked state with certainty [2]

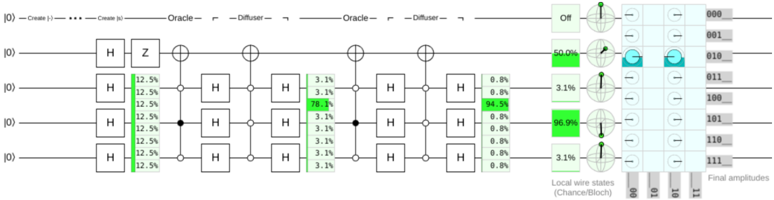
---

[1] G. L. Long. Grover Algorithm with zero theoretical failure rate, https://arxiv.org/pdf/quant-ph/0106071

Comparison with QUIRK

- Finds exact same probability

numQubits	ProbCorrect(SML)	ProbCorrect(Quirk)	ProbWrong(SML)	ProbWrong(Quirk)
1	0.5	0.5	0.5	0.5
2	1	1	0	0
3	0.9453125	0.9453125	0.0078125	0.0078125
4	0.908447265625	0.908447265625	0.006103515625	0.006103515625
5	0.896936535835	0.896936535835	0.00332462787628	0.00332462787628



Eval vs. Interp

- Differences
- Strengths

numQubits	ProbCorrect(Interp)	ProbCorrect(Eval)	ProbWrong(Interp)	ProbWrong(Eval)
1	0.5	0.5	0.5	0.5
2	1	1	0	0
3	0.9453125	0.9453125	0.0078125	0.0078125
4	0.908447265625	0.908447265625	0.006103515625	0.006103515625
5	0.896936535835	0.896936535835	0.00332462787628	0.00332462787628
6	0.816377019397	0.816377019397	0.00291465048576	0.00291465048576
7	0.683735462787	0.683735462787	0.00249027194656	0.00249027194656

Evaluation

- Results
- Extension of SML framework
- Building blocks
- Generality

Future Work

- Implementing an Oracle for the SAT Problem
- Translate higher-order oracle functions to quantum circuits
- Extending Support for Multiple Ancilla Bits
- Grovers Algorithm with zero theoretical failure rate
- Optimisation of Algorithm Circuits
- Implementing Additional Quantum Algorithms
- Comparison with Other Simulations

Conclusion

- Successfully implemented and simulated Grover's algorithm in SML
- Validated results using theoretical expectations and tools like Quirk
- Achieved high accuracy in small-scale simulations, with scaling challenges noted
- Modular implementation allows extension to other quantum algorithms
- Demonstrates potential for advancing hybrid quantum-classical computing