Vers la programmation des algorithmes quantiques

Benoît Valiron
CentraleSupélec – LRI, Univ. Paris Saclay

4 avril 2019

Big Picture: Quantum Computation

What **COULD** quantum algorithms be good for?

- factoring
  - for breaking modern cryptography
- simulating quantum systems
  - for more efficient molecule distillation procedure
- solving linear systems
  - for high-performance computing
- solving optimization problems
  - for big learning
- ... more than 300 algorithms:
  `http://math.nist.gov/quantum/zoo/`

Big Picture: Quantum Computation

Dichotomy between

- Quantum algorithms as theoretical tools for complexity analysis
- Quantum algorithms as practical tools for concrete problems

Challenges, assuming that a physical machine is available

- Designing the right computational model
- Moving from mathematical representation to code
- Resource estimation, optimization
- Compilation and low-level representation
- Debugging/unit testing hard : code analysis and verification

Plan

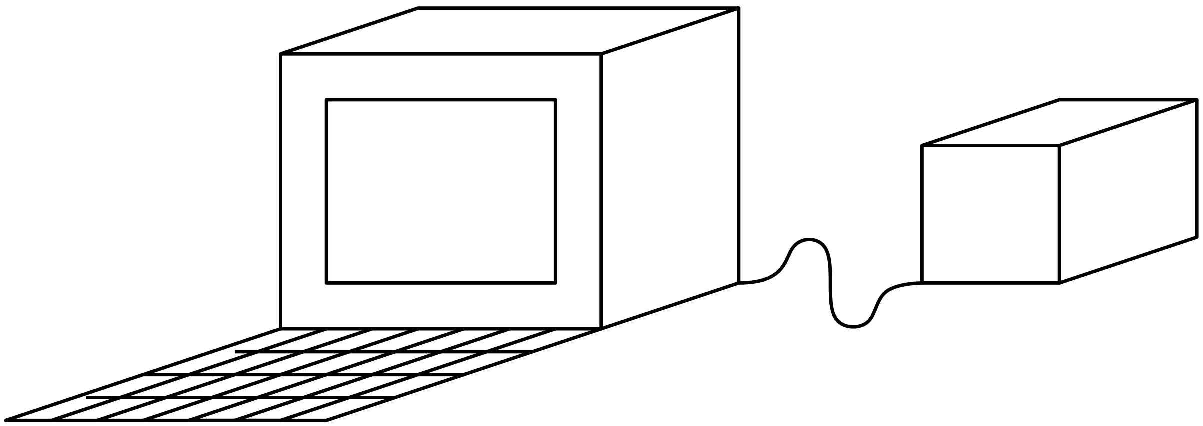
1. Computational Model
2. Internals of Algorithms
3. Coding Quantum Algorithms
4. A Language: Quipper
5. Example
6. Discussion

Plan

1. Computational Model
2. Internals of Algorithms
3. Coding Quantum Algorithms
4. A Language: Quipper
5. Example
6. Discussion

Computational Model

Classical unit = regular computer
Communicates with the coprocessor



Quantum unit = blackbox
Contains a quantum memory

Getting faster algorithms for conventional problems

Computational Model

A quantum memory with n quantum bits is a complex combination of strings of n bits. E.g. for $n = 3$:

$$\begin{array}{rl} & \alpha_0 \cdot \mathsf{000} \\ + & \alpha_1 \cdot \mathsf{001} \\ + & \alpha_2 \cdot \mathsf{010} \\ + & \alpha_3 \cdot \mathsf{011} \\ + & \alpha_4 \cdot \mathsf{100} \\ + & \alpha_5 \cdot \mathsf{101} \\ + & \alpha_6 \cdot \mathsf{110} \\ + & \alpha_7 \cdot \mathsf{111} \end{array}$$

with $|\alpha_0|^2 + |\alpha_1|^2 + |\alpha_2|^2 + |\alpha_3|^2 + |\alpha_4|^2 + |\alpha_5|^2 + |\alpha_6|^2 + |\alpha_7|^2 = 1$.

(alike probabilities with complex numbers. . .)

Computational Model

The operation one can perform on the memory are of three kinds:

1. Initialization/creation of a new quantum bit in a given *state*:

$$\begin{array}{rl} & \alpha_0 \cdot \mathtt{00} \\ + & \alpha_1 \cdot \mathtt{01} \\ + & \alpha_2 \cdot \mathtt{10} \\ + & \alpha_3 \cdot \mathtt{11} \end{array} \longmapsto \begin{array}{rl} & \alpha_0 \cdot \mathtt{000} \\ + & \alpha_1 \cdot \mathtt{010} \\ + & \alpha_2 \cdot \mathtt{100} \\ + & \alpha_3 \cdot \mathtt{110} \end{array}$$

Computational Model

The operation one can perform on the memory are of three kinds:

1. Initialization/creation of a new quantum bit in a given *state*:

$$\begin{array}{rl} & \alpha_0 \cdot \texttt{00} \\ + & \alpha_1 \cdot \texttt{01} \\ + & \alpha_2 \cdot \texttt{10} \\ + & \alpha_3 \cdot \texttt{11} \end{array} \longmapsto \begin{array}{rl} & \alpha_0 \cdot \texttt{001} \\ + & \alpha_1 \cdot \texttt{011} \\ + & \alpha_2 \cdot \texttt{101} \\ + & \alpha_3 \cdot \texttt{111} \end{array}$$

Computational Model

The operation one can perform on the memory are of three kinds:

2. Measurement. Measuring first qubit:

$$
\begin{array}{ll}
 & \alpha_0 \cdot 00 \\
+ & \alpha_1 \cdot 01 \\
 & \quad + \ \alpha_2 \cdot 10 \\
 & \quad + \ \alpha_3 \cdot 11
\end{array}
\longmapsto
\left\{
\begin{array}{lll}
 & \alpha_0 \cdot 00 & \\
+ & \alpha_1 \cdot 01 & (\text{prob. } |\alpha_0|^2 + |\alpha_1|^2) \\
 & & \\
 & \alpha_2 \cdot 10 & \\
+ & \alpha_3 \cdot 11 & (\text{prob. } |\alpha_2|^2 + |\alpha_3|^2)
\end{array}
\right.
$$

modulo renormalization.

Computational Model

The operation one can perform on the memory are of three kinds:

2. Measurement. Measuring second qubit:

$$\begin{array}{l} \alpha_0 \cdot 00 \\ \quad + \ \alpha_1 \cdot 01 \\ + \ \alpha_2 \cdot 10 \\ \quad + \ \alpha_3 \cdot 11 \end{array} \longmapsto \begin{cases} \alpha_0 \cdot 00 + \alpha_2 \cdot 10 & (\text{prob. } |\alpha_0|^2 + |\alpha_2|^2) \\ \alpha_1 \cdot 01 + \alpha_3 \cdot 11 & (\text{prob. } |\alpha_1|^2 + |\alpha_3|^2) \end{cases}$$

modulo renormalization.

Computational Model

The operation one can perform on the memory are of three kinds:

3. Unitary operations. Linear maps
   - preserving norms,
   - preserving orthogonality,
   - reversible.

   E.g. the N-gate on one quantum bit (flip). On the first qubit:

$$\begin{array}{rl} & \alpha_0 \cdot 00 \\ + & \alpha_1 \cdot 01 \\ + & \alpha_2 \cdot 10 \\ + & \alpha_3 \cdot 11 \end{array} \longmapsto \begin{array}{rl} & \alpha_0 \cdot 10 \\ + & \alpha_1 \cdot 11 \\ + & \alpha_2 \cdot 00 \\ + & \alpha_3 \cdot 01 \end{array}$$

Computational Model

The operation one can perform on the memory are of three kinds:

3. Unitary operations.

   E.g. the Hadamard gate on one quantum bit. Sends

$$\begin{array}{cccccc} 0 & \longmapsto & \frac{\sqrt{2}}{2} \cdot 0 & + & \frac{\sqrt{2}}{2} \cdot 1 \\ 1 & \longmapsto & \frac{\sqrt{2}}{2} \cdot 0 & - & \frac{\sqrt{2}}{2} \cdot 1 \end{array}$$

   When applied on the first qubit:

$$\begin{array}{rl} & \alpha_0 \cdot 01 \\ + & \alpha_1 \cdot 10 \end{array} \longmapsto \begin{array}{rl} & \alpha_0 \cdot (\frac{\sqrt{2}}{2} \cdot 01 + \frac{\sqrt{2}}{2} \cdot 11) \\ + & \alpha_1 \cdot (\frac{\sqrt{2}}{2} \cdot 00 - \frac{\sqrt{2}}{2} \cdot 10) \end{array}$$

Computational Model

The operation one can perform on the memory are of three kinds:

3. Unitary operations.

   E.g. the Hadamard gate on one quantum bit. Sends

$$
\begin{array}{ccccc}
0 & \longmapsto & \frac{\sqrt{2}}{2}\cdot 0 & + & \frac{\sqrt{2}}{2}\cdot 1 \\
1 & \longmapsto & \frac{\sqrt{2}}{2}\cdot 0 & - & \frac{\sqrt{2}}{2}\cdot 1
\end{array}
$$

   When applied on the first qubit:

$$
\begin{array}{rl}
& \alpha_0 \cdot 01 \\
+ & \alpha_1 \cdot 10
\end{array}
\longmapsto
\begin{array}{rl}
& \alpha_0 \frac{\sqrt{2}}{2} \cdot 01 \\
+ & \alpha_0 \frac{\sqrt{2}}{2} \cdot 11 \\
+ & \alpha_1 \frac{\sqrt{2}}{2} \cdot 00 \\
+ & \alpha_1 \frac{\sqrt{2}}{2} \cdot 10
\end{array}
$$

Computational Model

The operation one can perform on the memory are of three kinds:

3. Unitary operations.

   They can create superposition...

$$\mathtt{1100} \longmapsto \begin{array}{rl} & \frac{\sqrt{2}}{2} \cdot \mathtt{1100} \\ + & \frac{\sqrt{2}}{2} \cdot \mathtt{1110} \end{array}$$

   ...or remove it

$$\begin{array}{rl} & \frac{\sqrt{2}}{2} \cdot \mathtt{1100} \\ + & \frac{\sqrt{2}}{2} \cdot \mathtt{1110} \end{array} \longmapsto \mathtt{1100}$$

Computational Model

The operation one can perform on the memory are of three kinds:

3. Unitary operations.

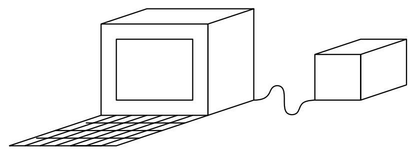
   They can simulate classical operations:

   - Bit-flip (N-gate).
   - Tests (Controlled operations). E.g. Controlled-not. Second qubit is controlling:

$$
\begin{array}{rl} & \alpha_0 \cdot 00 \\ + & \alpha_1 \cdot 01 \\ + & \alpha_2 \cdot 10 \\ + & \alpha_3 \cdot 11 \end{array}
\longmapsto
\begin{array}{rl} & \alpha_0 \cdot 00 \\ + & \alpha_1 \cdot 11 \\ + & \alpha_2 \cdot 10 \\ + & \alpha_3 \cdot 01 \end{array}
=
\begin{array}{rl} & \alpha_0 \cdot 00 \\ + & \alpha_3 \cdot 01 \\ + & \alpha_2 \cdot 10 \\ + & \alpha_1 \cdot 11 \end{array}
$$

Computational Model

A co-processor with an internal (quantum) memory



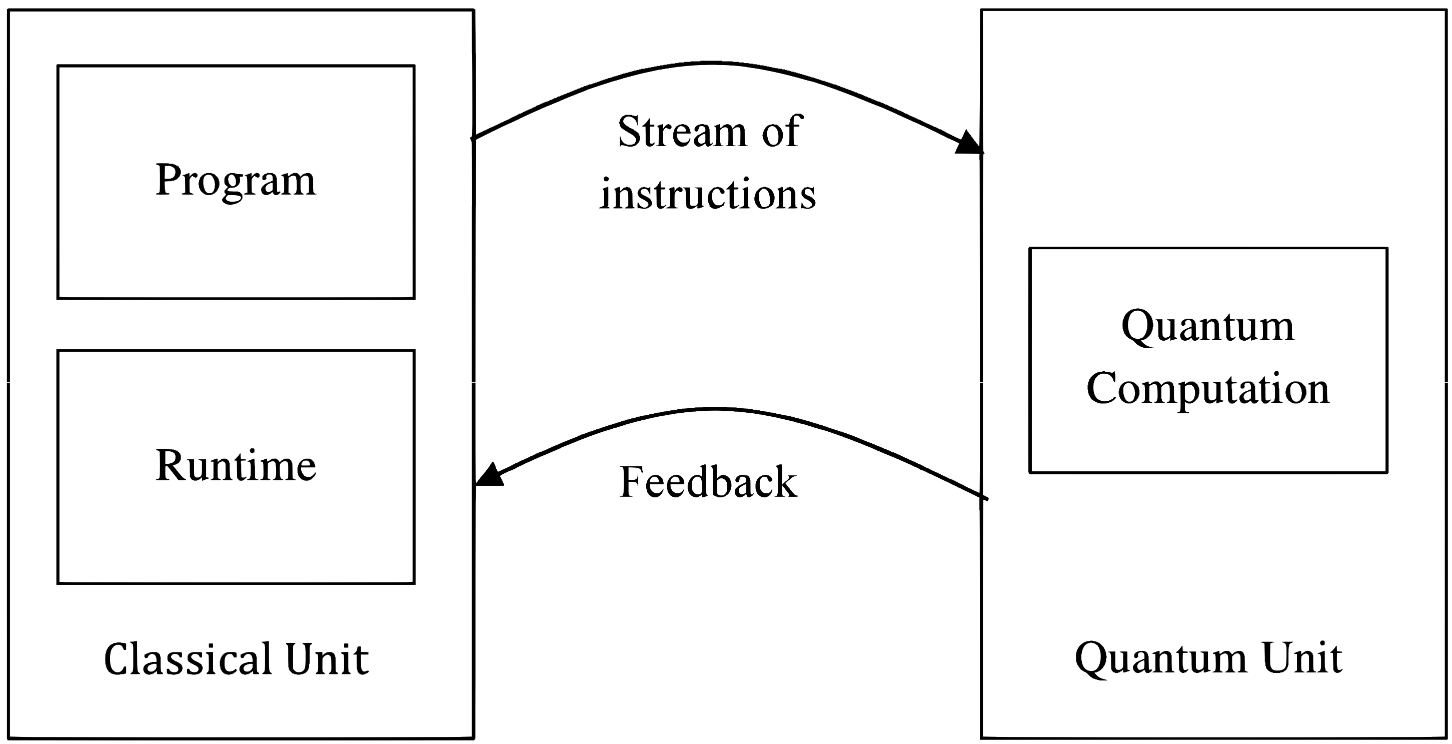
- A random access model.
  $\rightarrow$ for each qubit: Alloc/init, unitary operations, measurements
- Specific I/O interface
- Measurement triggers a probablistic side-effect

To note

- Classical data can transparently flow in.
- To act on quantum memory, classical operations have to lifted.
- Local actions on one (or two) qubit(s) at a time
- Limited moving of qubits; no copying

Computational Model

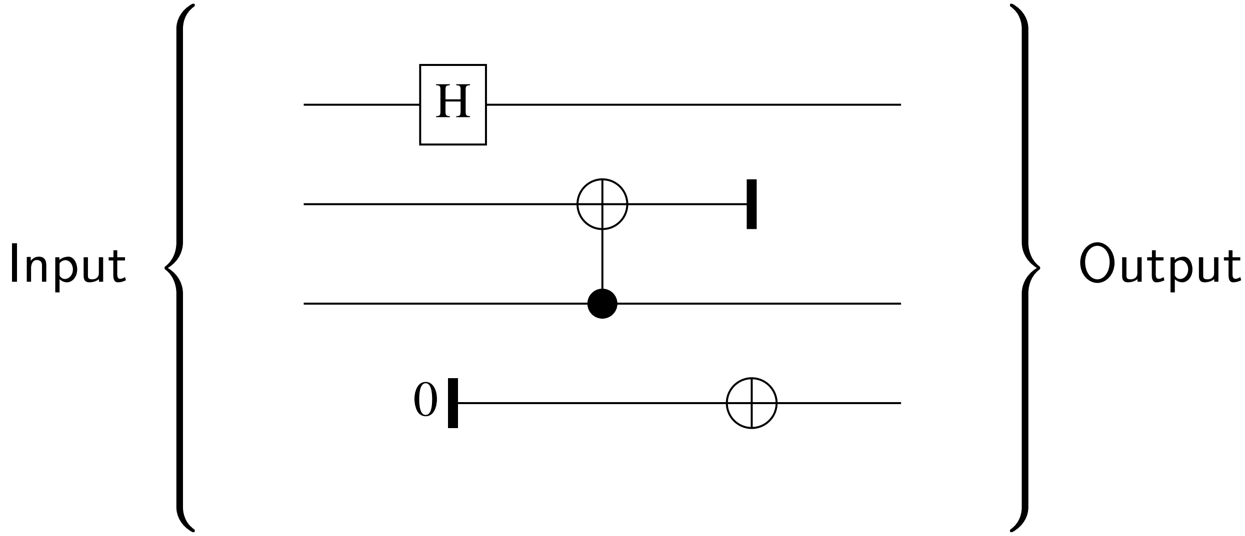
Typical execution flow



Computational Model

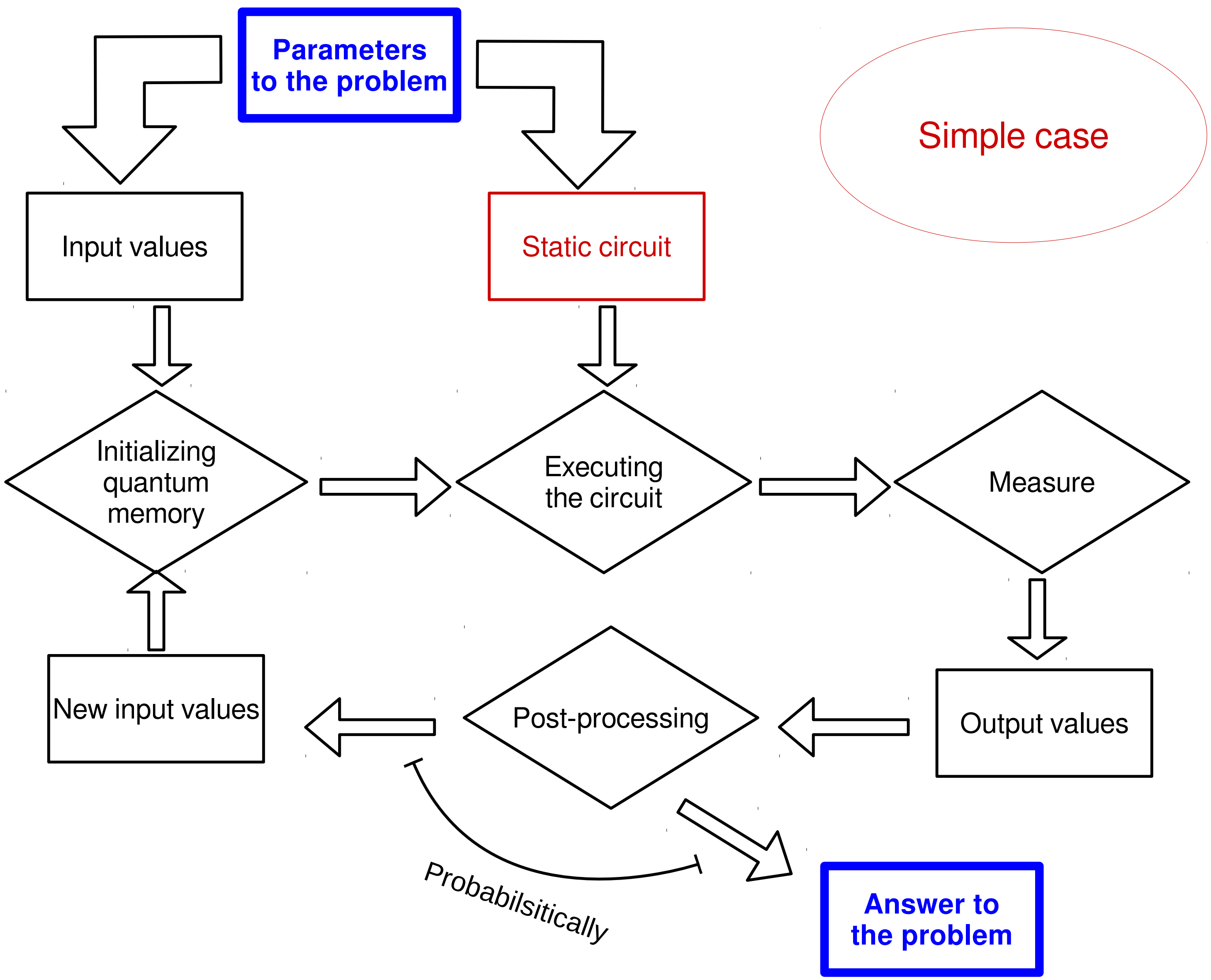
Stream of instructions

- Series of elementary actions applied on the quantum memory
- Input/Output of actions summarized with a quantum circuit.
- wire $\equiv$ qubit, box $\equiv$ action, time flows left-to-right



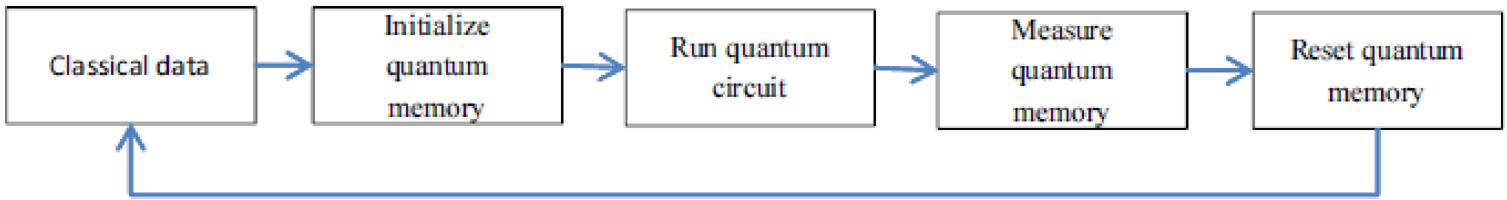
No "quantum loop" or "conditional escape".

Computational Model

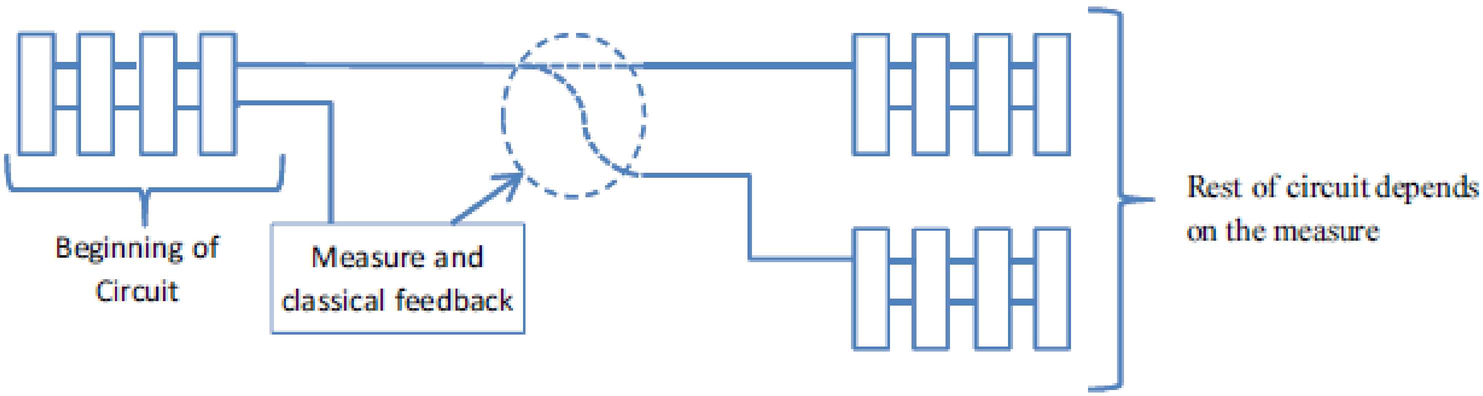


Computational Model

Some algorithms follow a simple scheme



Others are following a more adaptative scheme:



This is where quantum circuits differ from hardware design.

One cannot draw a quantum circuit once and for all.

Computational Model

A sound model of computation:

Interaction with the quantum memory seen as an I/O side effect

```
Circ a := Empty a
        | Write Gate (Circ a)
        | Read Wire (Bool -> (Circ a))
```

- Output: emit gates to the co-processor
- Input: emit a read even to the co-processor, with a call-back function

Representing circuits

- static circuits: lists of gates
- dynamic circuits: trees of gates.

Computational Model

Moral

- Distinction parameter / input
- Circuits might be dynamically generated
- Parameters = govern the shape and size of the circuit
- Model of computation : two side-effects:
  - "quantum I/O" : state and I/O
  - probability

Plan

1. Computational Model
2. Internals of Algorithms
3. Coding Quantum Algorithms
4. A Language: Quipper
5. Example
6. Discussion

Internals of algorithms

The techniques used to described quantum algorithms are diverse.

1. Quantum primitives
   - Phase estimation.
   - Amplitude amplification.
   - Quantum walk.

Should come up as a programmable library

Internals of algorithms

The techniques used to described quantum algorithms are diverse.

2. Oracles.
   - Take a classical function $f : \texttt{Bool}^n \to \texttt{Bool}^m$.
   - Construct

$$\begin{array}{rccc} \overline{f}: & \texttt{Bool}^{n+m} & \longrightarrow & \texttt{Bool}^{n+m} \\ & (x,y) & \longmapsto & (x, y \oplus f(x)) \end{array}$$

   - Build the unitary U_f acting on $n+m$ qubits computing $\overline{f}$.

Internals of algorithms

The techniques used to described quantum algorithms are diverse.

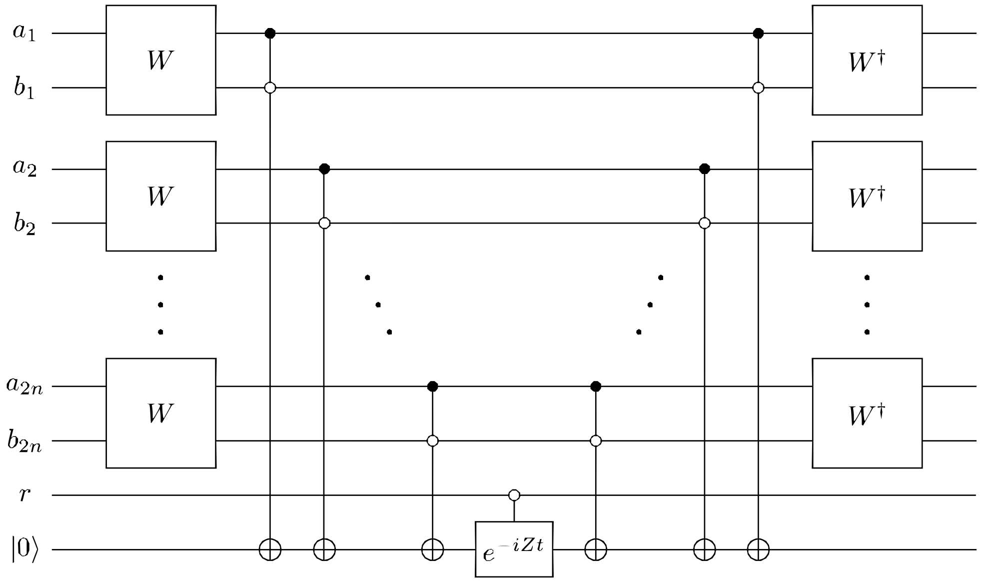
2. Oracles, in real life

```
calcRweights y nx ny lx ly k theta phi =
 let (xc',yc') = edgetoxy y nx ny in
 let xc = (xc'-1.0)*lx - ((fromIntegral nx)-1.0)*lx/2.0 in
 let yc = (yc'-1.0)*ly - ((fromIntegral ny)-1.0)*ly/2.0 in
 let (xg,yg) = itoxy y nx ny in
 if (xg == nx) then
     let i = (mkPolar ly (k*xc*(cos phi)))*(mkPolar 1.0 (k*yc*(sin phi)))*
             ((sinc (k*ly*(sin phi)/2.0))+0.0) in
     let r = ( cos(phi)+k*lx )*((cos (theta - phi))/lx+0.0) in i*r
 else if (xg==2*nx-1) then
     let i = (mkPolar ly (k*xc*cos(phi)))*(mkPolar 1.0 (k*yc*sin(phi)))*
             ((sinc (k*ly*sin(phi)/2.0))+0.0) in
     let r = ( cos(phi)+(- k*lx))*((cos (theta - phi))/lx+0.0) in i*r
 else if ( (yg==1) and (xg<nx) ) then
     let i = (mkPolar lx (k*yc*sin(phi)))*(mkPolar 1.0 (k*xc*cos(phi)))*
             ((sinc (k*lx*(cos phi)/2.0))+0.0) in
     let r = ( (- sin phi)+k*ly )*((cos(theta - phi))/ly+0.0) in i*r
 else if ( (yg==ny) and (xg<nx) ) then
     let i = (mkPolar lx (k*yc*sin(phi)))*(mkPolar 1.0 (k*xc*cos(phi)))*
             ((sinc (k*lx*(cos phi)/2.0))+0.0) in
     let r = ( (- sin phi)+(- k*ly) )*((cos(theta - phi)/ly)+0.0) in i*r
 else 0.0+0.0
```

Internals of algorithms

The techniques used to described quantum algorithms are diverse.

3. Blocks of loosely-defined low-level circuits.



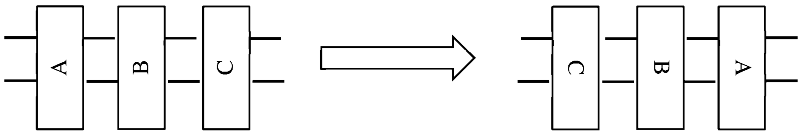
- This is not a formal specification!
- Notion of "box"
- Size of the circuit depends on parameters

Internals of algorithms

The techniques used to described quantum algorithms are diverse.

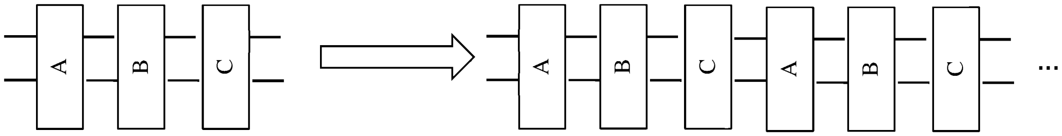
4. High-level operations on circuit:

- Circuit inversion.



(the circuit needs to be reversible. . .)

- Repetition of the same circuit.



(needs to have the same input and output arity. . .)

- Controlling of circuits

Internals of algorithms

The techniques used to described quantum algorithms are diverse.

5. Classical processing.
   - Generating the circuit...
   - Computing the input to the circuit.
   - Processing classical feedback in the middle of the computation.
   - Analyzing the final answer (and possibly starting over).

Internals of algorithms

Summary

- Need of automation for oracle generation
- Distinction parameter / input
- Circuits as inputs to other circuits
- Regularity with respect to the size of the input
- Circuit construction:
  - Using circuit combinators: Inversion, repetition, control, *etc*
  - Procedural
- Lots of classical processing!

Plan

1. Computational Model
2. Internals of Algorithms
3. Coding Quantum Algorithms
4. A Language: Quipper
5. Example
6. Discussion

Coding algorithms

A very recent topic

- From complexity analysis to concrete circuits
- No machine yet, but
  - Resource analysis
  - Optimization
  - Verification
- Scalable languages: in the last 5 years
  - Python's libraries/DSL: Project-Q, QISKit, *etc*
  - Liqui$|\rangle$, Q# (Microsoft)
  - Quipper, QWIRE (academic)
  - ...

Coding algorithms

Imperative programming and the quantum I/O

- Quantum I/O: using commands
- Measurement: returns a boolean (probabilistically)
- If well-behaved, provides high-level circuit operations
- Example with Project-Q (Simplified)

```
def circuit(q1,q2,q3):
  H | q1
  with Control(q1):
    X | q2
    H | q3
  x = Measure | q1
  if x:
    Y | q2
  else:
    Z | q2
```

Coding algorithms

Imperative programming and the quantum I/O

- Quantum I/O: using commands
- Measurement: returns a boolean (probabilistically)
- If well-behaved, provides high-level circuit operations
- Example with Project-Q (Simplified)

```
def circuit(q1,q2,q3):
  H | q1
  with Control(q1):
    X | q2
    H | q3
  x = Measure | q1
  if x:
    Y | q2
  else:
    Z | q2
```

Coding algorithms

Functional programming and the quantum I/O

- Monadic approach to encapsulate I/O
- Inside the monad: quantum operations
- Outside the monad: classical operations and circuit manipulation
- Qubits only live inside the monad

Coding algorithms

Dealing with run-time errors

- Imperative-style: Quantum I/O is a memory mapping
  - $\rightarrow$ Type-systems based on separation logic should work
  - Hoare logic or Contracts
- Functional-style:
  - Non-duplicable quantum data: linear type system
  - Dependent-types

Plan

1. Computational Model
2. Internals of Algorithms
3. Coding Quantum Algorithms
4. A Language: Quipper
5. Example
6. Discussion

A Language: Quipper

- Embedded language in Haskell
- Logical description of hierarchical circuits
- Well-founded monadic semantics. Allow to mix two paradigms
  - Procedural : describing low-level circuits
  - Declarative : describing high-level operation
- Parameter/input distinction
  - Parameter : determine the shape of the circuit
  - Input : determine what goes in the wires
- $\cdots$

A Language: Quipper

A function in Quipper is a map

`A -> Circ B`

- Input something of type `A`
- Output something of type `B`
- As a side effect, generate a circuit snippet

Or

- Input a value of type `A`
- Output a "computation" of type `Circ B`

Families of circuits

- represented with lists, e.g. `[Qubit] -> Circ [Qubit]`

A Language: Quipper

New base type : `Qubit` ≡ wire

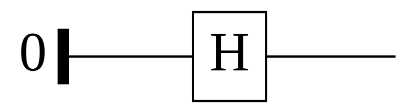
Building blocks

- `qinit :: Bool -> Circ Qubit`
- `qdiscard :: Qubit -> Circ ()`
- `hadamard :: Qubit -> Circ Qubit`
- `hadamard_at :: Qubit -> Circ ()`

Composition of functions ≡ composition of circuits

$$\texttt{Bool} \xrightarrow{\texttt{qinit}} \texttt{Circ Qubit}$$

$$\texttt{Qubit} \xrightarrow{\texttt{hadamard}} \texttt{Circ Qubit}$$

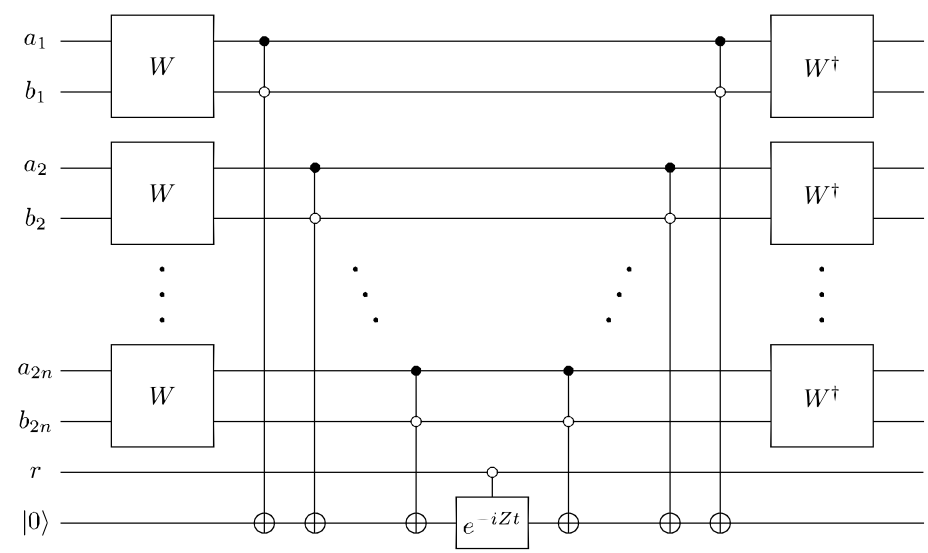


High-level circuit combinators

- `controlled :: Circ a -> Qubit -> Circ a`
- `inverse :: (a -> Circ b) -> b -> Circ a`

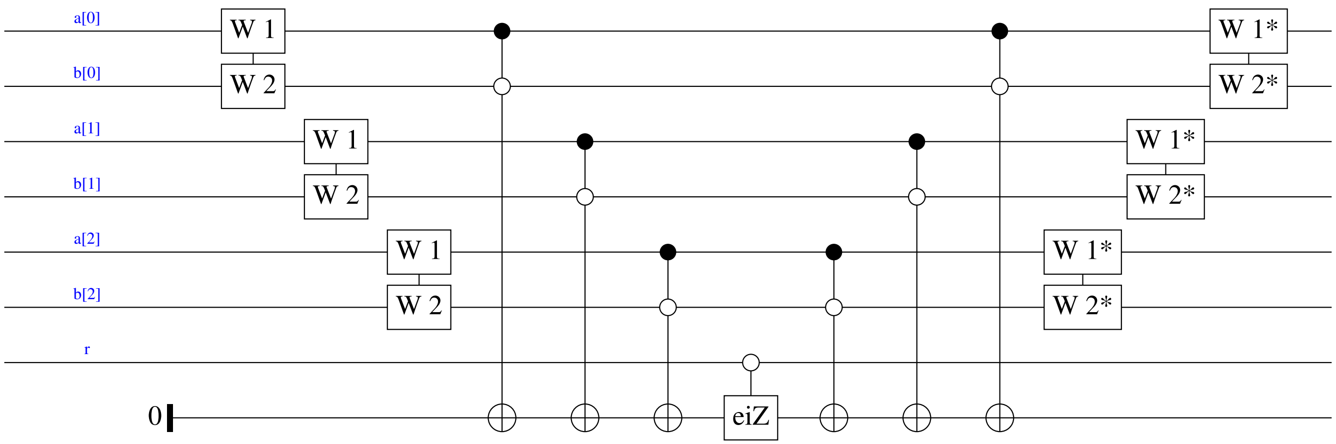
A Language: Quipper

```
import Quipper

w :: (Qubit,Qubit) -> Circ (Qubit,Qubit)
w = named_gate "W"

toffoli :: Qubit -> (Qubit,Qubit) -> Circ Qubit
toffoli d (x,y) =
  qnot d `controlled` x .==. 1 .&&. y .==. 0

eiz_at :: Qubit -> Qubit -> Circ ()
eiz_at d r =
  named_gate_at "eiZ" d `controlled` r .==. 0

circ :: [(Qubit,Qubit)] -> Qubit -> Circ ()
circ ws r = do
  label (unzip ws,r) (("a","b"),"r")
  d <- qinit 0
  mapM_ w ws
  mapM_ (toffoli d) ws
  eiz_at d r
  mapM_ (toffoli d) (reverse ws)
  mapM_ (reverse_generic w) (reverse ws)
  return ()

main = print_generic EPS circ (replicate 3 (qubit,qubit)) qubit
```



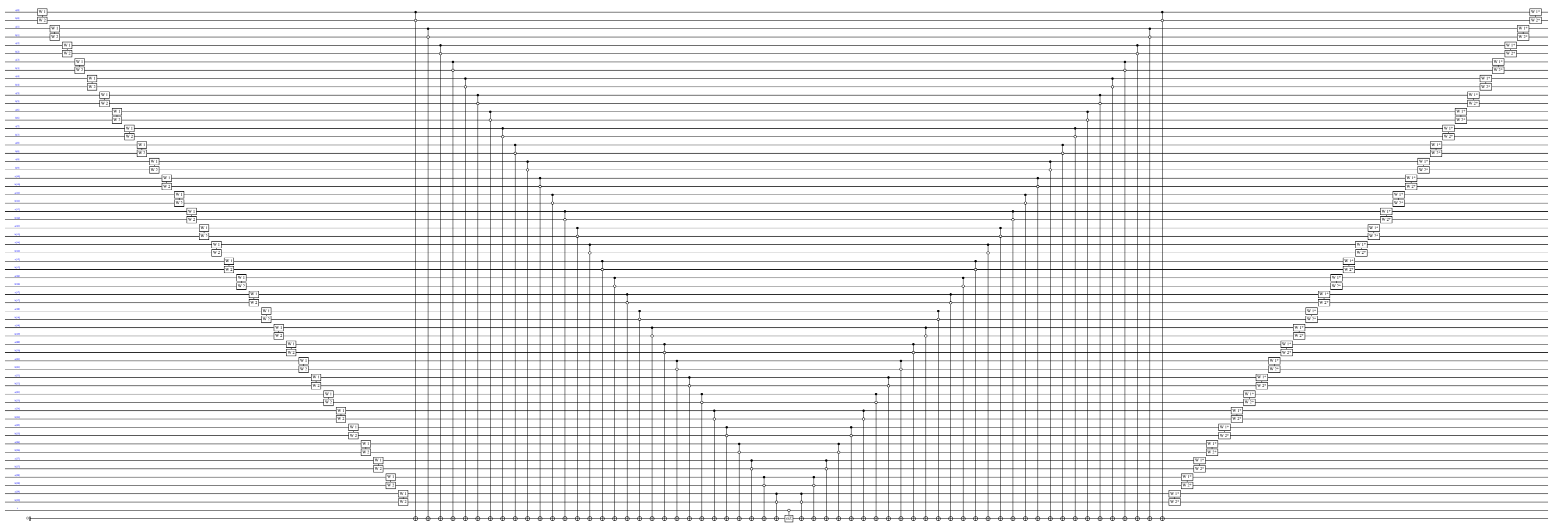
A Language: Quipper

Result (3 wires):



A Language: Quipper

Result (30 wires):

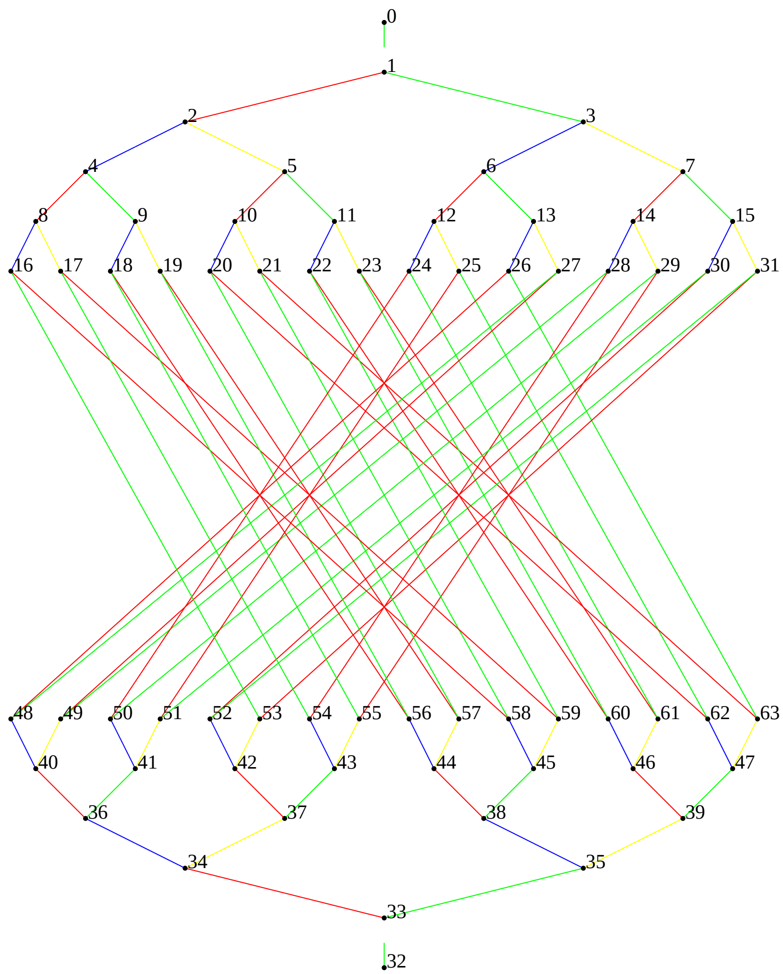


Plan

1. Computational Model
2. Internals of Algorithms
3. Coding Quantum Algorithms
4. A Language: Quipper
5. Example
6. Discussion

Concrete example: BWT

Binary Welded Tree Algorithm



- Start at entrance, look for exit
- Description of the graph:

$$I \;:\; \texttt{Node}$$

$$G \;:\; \texttt{Color} \times \texttt{Node} \to \texttt{Maybe Node}$$

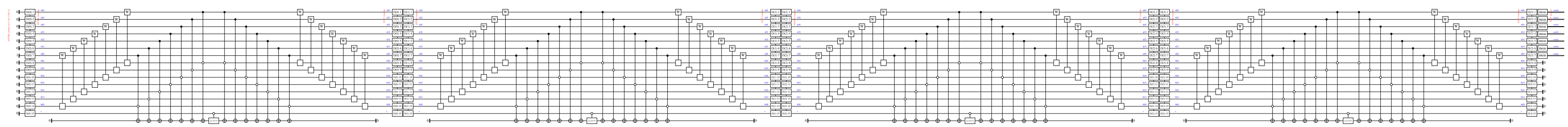
$$O \;:\; \texttt{Node} \to \texttt{Bool}$$

- Random/Quantum walk
- Parameters:
  height of tree ; number of steps.

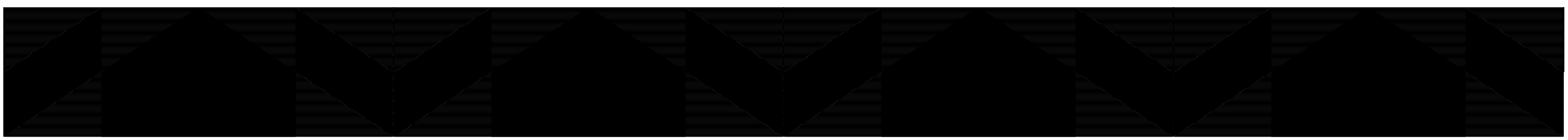
Concrete example: BWT

Using Quipper, w/o oracle:

```
$ ./bwt -o blackbox -n 5 -s 1 -f PDF
```



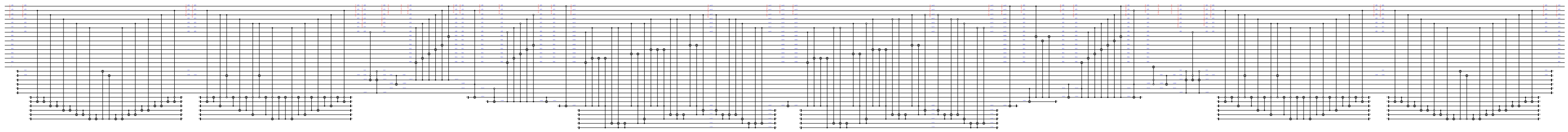
```
$ ./bwt -o blackbox -n 300 -s 1 -f PDF
```



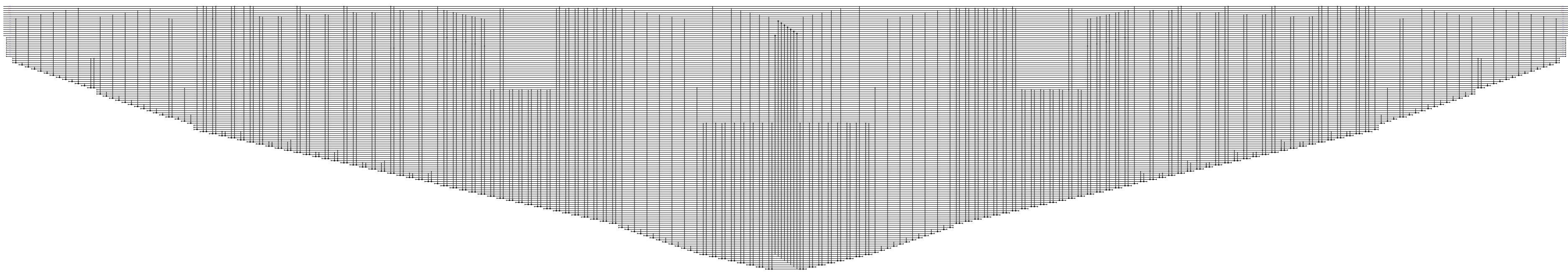
Concrete example: BWT

Using Quipper, the oracles:

```
$ ./bwt -o orthodox -O -n 5 -s 1 -f PDF
```



```
$ time ./bwt -o template -O -n 5 -s 1 -f PDF
```



Plan

1. Computational Model
2. Internals of Algorithms
3. Coding Quantum Algorithms
4. A Language: Quipper
5. Example
6. Discussion

Discussion

- Big-O and concrete resource estimation
- Towards program analysis

Big-O: Case of the QLS algorithm

Quantum Linear System

Considering a vector $\vec{b}$ and the system

$$A \cdot \vec{x} = \vec{b},$$

compute the value of $\langle \vec{x} \,|\, \vec{r} \rangle$ for some vector $\vec{r}$.

Practical situation: the matrix A corresponds to the finite-element approximation of the scattering problem.

(`arXiv:1505.06552`, based on Clader *et al*, 2013)

Big-O: Case of the QLS algorithm

Three oracles:

- for $\vec{r}$ and for $\vec{b}$: input an index, output (the representation of) a complex number
- for A: input two indices, output also a complex number

Many quantum primitives

- Amplitude estimation
- Phase estimation
- Amplitude amplification
- Hamiltonian simulation

In Quipper

- $\sim$ 3000 lines of code

Big-O: Case of the QLS algorithm

The parameters are

κ: condition number (large) d: sparseness of the matrix

N: size of the matrix (large) ϵ: desired max error (small)

In the litterature, the number of gates:

Harrow *et al* (2009) $\tilde{O}(\kappa^2 d^2 \log(N)/\epsilon)$

Clader *et al* (2013) $\tilde{O}(\kappa d^4 \log(N)/\epsilon^2)$

`arXiv:1505.06552`: $\kappa = 10^4$, $d = 7$, $N = 332,020,680$, $\epsilon = 10^{-2}$.

The big-O: $\sim 10^{12}$

Big-O: Case of the QLS algorithm

The parameters are

κ: condition number (large) d: sparseness of the matrix

N: size of the matrix (large) ϵ: desired max error (small)

In the litterature, the number of gates:

Harrow *et al* (2009) $\tilde{O}(\kappa^2 d^2 \log(N)/\epsilon)$

Clader *et al* (2013) $\tilde{O}(\kappa d^4 \log(N)/\epsilon^2)$

arXiv:1505.06552: $\kappa = 10^4$, $d = 7$, $N = 332,020,680$, $\epsilon = 10^{-2}$.

The big-O: $\sim 10^{12}$

Careful counting: $\sim 10^{29}$

Towards tools for program analysis

One cannot "read" the quantum memory

- Testing / debugging expensive
- Probabilistic model
- What does it mean to have a "correct" implementation?

Emulation of circuits

- Only for "small" instances
- Taming the testing problem
- For experimentation of error models

Formal methods

- Type systems: capture errors at compile-times
- Static analyis tools:

  analyze and resource estimation for quantum programs
- Proof assistants: verify code transformation and optimization