Methods for Processing Quantum Circuits

from the context of the IBM Q Hub

Dr. Wolfgang Alexander Gehrke

Universität der Bundeswehr München
Research Institute CODE

Munich, November 11, 2020

Forschungsinstitut
Cyber Defence
Universität der Bundeswehr München

Contents

Forschungsinstitut
Cyber Defence
Universität der Bundeswehr München

Some Background

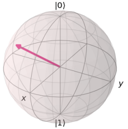
- ingredients for quantum computing

  complex numbers $|\psi\rangle = \binom{\alpha}{\beta} = \alpha\binom{1}{0} + \beta\binom{0}{1} = \alpha\,|0\rangle + \beta\,|1\rangle$

  probability theory $P(M(|\psi\rangle)) = \begin{cases} 0 = |\alpha|^2 \\ 1 = |\beta|^2 \end{cases} \quad |\alpha|^2 + |\beta|^2 = 1$

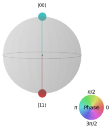
  linear algebra $|\psi\rangle \in \mathbb{C}^2, \otimes : \mathbb{H}_m \times \mathbb{H}_n \to \mathbb{H}_{mn}, \langle Uv|Uw\rangle = \langle v|w\rangle$
- visualizations $|\psi\rangle = e^{-\imath\phi/2}\sin\frac{\theta}{2}\,|0\rangle + e^{\imath\phi/2}\cos\frac{\theta}{2}\,|1\rangle$

Bloch sphere



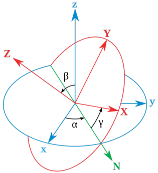
superposition

Qsphere



entanglement

1Qubit-operations



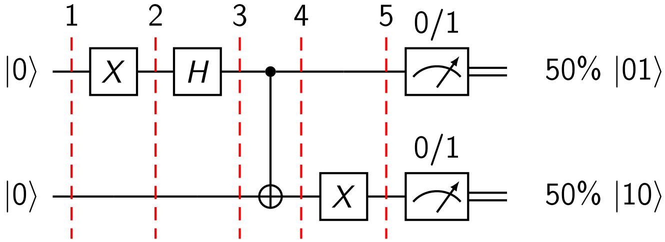
rotation

n Qubits

require $\mathbb{C}^{2^n}$

Forschungsinstitut Cyber Defence
Universität der Bundeswehr München

The Circuit Model



1	2	3	4	5
$\begin{pmatrix}1\\0\\0\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\1\\0\end{pmatrix}$	$\begin{pmatrix}\frac{1}{\sqrt{2}}\\0\\\frac{-1}{\sqrt{2}}\\0\end{pmatrix}$	$\begin{pmatrix}\frac{1}{\sqrt{2}}\\0\\0\\\frac{-1}{\sqrt{2}}\end{pmatrix}$	$\begin{pmatrix}0\\\frac{1}{\sqrt{2}}\\\frac{-1}{\sqrt{2}}\\0\end{pmatrix}$
$\vert 00\rangle$	$\vert 10\rangle$	$\frac{1}{\sqrt{2}}(\vert 00\rangle - \vert 10\rangle)$	$\frac{1}{\sqrt{2}}(\vert 00\rangle - \vert 11\rangle)$	$\frac{1}{\sqrt{2}}(\vert 01\rangle - \vert 10\rangle)$

Forschungsinstitut Cyber Defence Universität der Bundeswehr München

The Theory

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \quad S = \begin{pmatrix}1 & 0\\ 0 & \imath\end{pmatrix} \quad T = \begin{pmatrix}1 & 0\\ 0 & e^{\frac{\imath\pi}{4}}\end{pmatrix} \sim \begin{pmatrix}e^{\frac{-\imath\pi}{8}} & 0\\ 0 & \frac{\imath\pi}{8}\end{pmatrix}$$

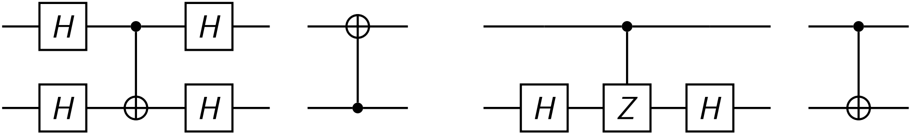
$$CX = |00\rangle\langle 00| + |01\rangle\langle 01| + |11\rangle\langle 10| + |10\rangle\langle 11| : |xy\rangle \to |x(x \oplus_2 y)\rangle$$

universal quantum gate set $\{CX, U\}$ i.e. CNOT with all 1Qubit-ops

Gottesman-Knill theorem $\{CX, H, S\}$ can be *effectively* simulated

Solovay–Kitaev theorem $\{CX, H, T\}$ can *efficiently* approximate

$$U(\theta,\phi,\lambda) = R_z(\phi)R_y(\theta)R_z(\lambda) = \begin{pmatrix}e^{-\imath(\phi+\lambda)/2}\cos\frac{\theta}{2} & -e^{-\imath(\phi-\lambda)/2}\sin\frac{\theta}{2}\\ e^{\imath(\phi-\lambda)/2}\sin\frac{\theta}{2} & e^{\imath(\phi+\lambda)/2}\cos\frac{\theta}{2}\end{pmatrix}$$

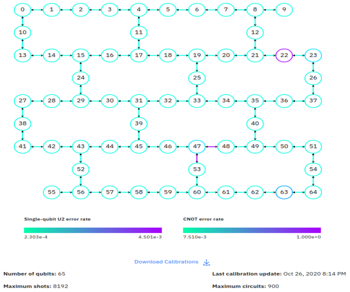


Forschungsinstitut Cyber Defence
Universität der Bundeswehr München

$$SWAP = CX\downarrow \circ CX\uparrow \circ CX\downarrow = CX\uparrow \circ CX\downarrow \circ CX\uparrow$$

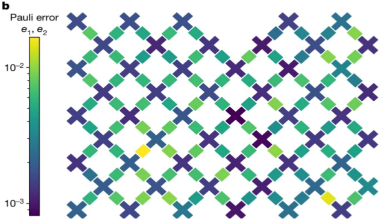
Real Devices

IBM 2020



Google 2019

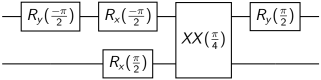
Average error	Isolated	Simultaneous
Single-qubit (e_1)	0.15%	0.16%
Two-qubit (e_2)	0.36%	0.62%
Two-qubit, cycle (e_{2c})	0.65%	0.93%
Readout (e_r)	3.1%	3.8%



AQT (ion-trap ca. 50 Qubits):

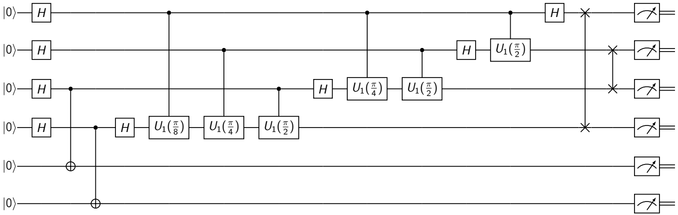
- 2Q Ops: Ising-$XX(\frac{\pi}{2n})$ gate
- connectivity: all-to-all
- fidelity: ~1Q:99.7% 2Q:99.5%

CNOT↓ equivalent:



FI CODE Forschungsinstitut Cyber Defence Universität der Bundeswehr München

A Language for Circuits




```
OPENQASM 2.0;
include "qelib1.inc";
qreg qx[4];
qreg qy[2];
creg mx[4];
creg my[2];
gate mod4 a,b,c,d,e,f { cx c,e; cx d,f; }
gate qft4 q0,q1,q2,q3 {
  h q3; cu1(pi/8) q0,q3; cu1(pi/4) q1,q3; cu1(pi/2) q2,q3;
  h q2; cu1(pi/4) q0,q2; cu1(pi/2) q1,q2;
  h q1; cu1(pi/2) q0,q1; h q0; swap q0,q3; swap q1,q2; }
h qx;
mod4 qx[0],qx[1],qx[2],qx[3],qy[0],qy[1];
qft4 qx[0],qx[1],qx[2],qx[3];
measure qx -> mx;
measure qy -> my;
```

```
// file: qelib1.inc
gate u3(theta,phi,lambda) q
     { U(theta,phi,lambda) q; }
gate u2(phi,lambda) q { U(pi/2,phi,lambda) q; }
gate u1(lambda) q { U(0,0,lambda) q; }
gate cx c,t { CX c,t; }
gate id a { U(0,0,0) a; }
gate x a { u3(pi,0,pi) a; }
gate y a { u3(pi,pi/2,pi/2) a; }
gate z a { u1(pi) a; }gate h a { u2(0,pi) a; }
gate s a {u1(pi/2) a;}gate sdg a {u1(-pi/2) a;}
gate t a {u1(pi/4) a;}gate tdg a {u1(-pi/4) a;}
gate ccx a,b,c {h c; cx b,c; tdg c; cx a,c;
  t c; cx b,c; tdg c; cx a,c;
  t b; t c; h c; cx a,b; t a;
  tdg b; cx a,b; }
```

A Grammar

```
<mainprogram> |= OPENQASM <real> ; <program>
<program>     |= <statement>
              |  <program> <statement>
<statement>   |= <decl>
              |  <gatedecl> <goplist> }
              |  <gatedecl> }
              |  opaque <id> <idlist> ;
              |  opaque <id> ( ) <idlist> ;
              |  opaque <id> ( <idlist> ) <idlist> ;
              |  <qop>
              |  if ( <id> == <nninteger> ) <qop>
              |  barrier <anylist> ;
<decl>        |= qreg <id> [ <nninteger> ] ;
              |  creg <id> [ <nninteger> ] ;
<gatedecl>    |= gate <id> <idlist> {
              |  gate <id> ( ) <idlist> {
              |  gate <id> ( <idlist> )<idlist> {
<goplist>     |= <uop>
              |  barrier <idlist> ;
              |  <goplist> <uop>
              |  <goplist> barrier <idlist> ;
```

```
<qop>       |= <uop>
            |  measure <argument> - > <argument> ;
            |  reset <argument> ;
<uop>       |= U ( <explist> ) <argument> ;
            |  CX <argument> , <argument> ;
            |  <id> <anylist> ;
            |  <id> ( ) <anylist> ;
            |  <id> ( <explist> ) <anylist> ;
<anylist>   |= <idlist> | <mixedlist>
<idlist>    |= <id> | <idlist> , <id>
<mixedlist> |= <id> [ <nninteger> ]
            |  <mixedlist> , <id>
            |  <mixedlist> , <id> [ <nninteger> ]
            |  <idlist> , <id>[ <nninteger> ]
<argument>  |= <id> | <id> [ <nninteger> ]
<explist>   |= <exp> | <explist> , <exp>
<exp>       |= <real> | <nninteger> | pi | <id>
            |  <exp> + <exp> | <exp> - <exp>
            |  <exp> * <exp> | <exp> / <exp>
            |  - <exp> | <exp> ^ <exp>
            |  ( <exp> ) | <unaryop> ( <exp> )
<unaryop>   |= sin | cos | tan | exp | ln | sqrt
```

```
id        :=  [a-z][A-Za-z0-9_]*
real      :=  ([0-9]+\.[0-9]*|[0-9]*\.[0-9]+)([eE][-+]?[0-9]+)?
nninteger :=  [1-9]+[0-9]*|0
```

YACC and LEX

ocamlyacc
: to implement the *grammar*

ocamllex
: to implement regular expressions for language *tokens*

goal
: construct *syntax tree* for manipulation

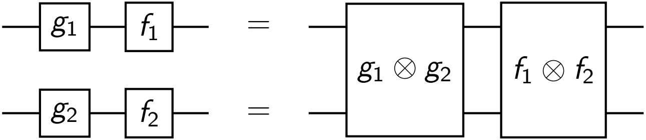
benefits
1. iteration unrolling
2. gate definition expansion
3. determine independent qubit sets
4. annotations for input/output/ancilla qubits
5. assertion checking like bounds of arrays
6. automatic uncomputation generation
7. circuit transformations

Pictures versus Equations

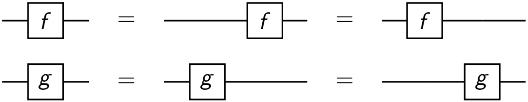
sequential composition $f \circ g$ f *after* g (matrix product)

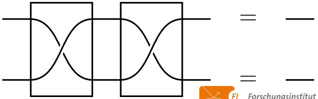
parallel composition $f \otimes g$ f *while* g (tensor product)

fundamental equation $(f_1 \circ g_1) \otimes (f_2 \circ g_2) = (f_1 \otimes f_2) \circ (g_1 \otimes g_2)$

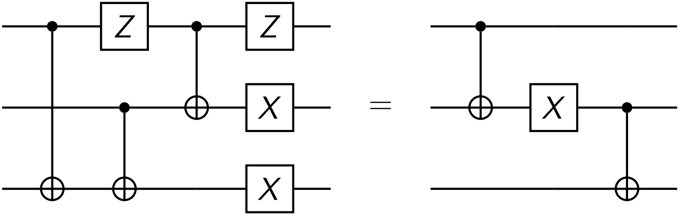


$f \otimes g = (f \circ I) \otimes (I \circ g) = (I \circ f) \otimes (g \circ I)$ $SWAP \circ SWAP = I \otimes I$





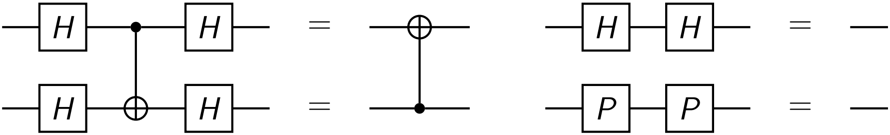
A Computer Algebra System

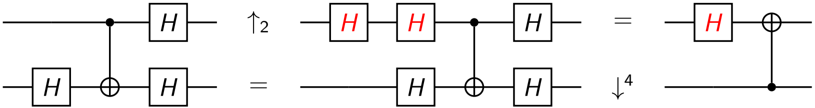



```
(1) -> px, py, pz, hd, hd * hd, swaps, cnotD
  (1)                          + +-+    +-+ +
                               |\|2    \|2  |        +1  0  0  0+ +1  0  0  0+
                               |----   ---- |        |          | |          |
    +0  1+ +0   - %i+ +1   0 + |  2      2  | +1  0+ |0  0  1  0| |0  1  0  0|
   [|    |,|        |,|      |,|            |,|    |,|          |,|          |]
    +1  0+ +%i   0  + +0  - 1+ | +-+     +-+| +0  1+ |0  1  0  0| |0  0  0  1|
                               |\|2     \|2 |        |          | |          |
                               |----  - ----|        +0  0  0  1+ +0  0  1  0+
                               +  2       2 +
Type: Tuple(Matrix(Expression(Complex(Integer))))
(2) -> ( kron(pz,kron(px,px)) * kron(cnotD,id) * kron(pz,cnotD) *
         kron(swaps,id) * kron(id,cnotD) * kron(swaps,id) =
         kron(id,cnotD) * kron(id,kron(px,id)) * kron(cnotD,id) ) :: Boolean
  (2)  true
Type: Boolean
(3) ->
```

"Critical Pairs"

similarities to *term rewriting*

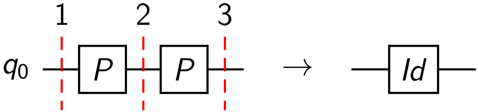




rules should *reduce* some measure (e.g. gate count)
hence here a new rule was found

Help from Logic Programming

$P \in \{H, X, Y, Z\}$




```
rule1 :-
    member(P, ["h", "x", "y", "z"]),
    wire(From, gate(P, [], _), Middle),
    wire(Middle, gate(P, [], _), To),
    retract(wire(From, gate(P, [], _), Middle)),
    retract(wire(Middle, gate(P, [], _), To)),
    From = [moment(Q,_)],
    assert(wire(From, gate("Id", [], [Q]), To)),
    writeln("Rule 1 fired").
```

```
try     :- rule3 ; rule2 ; rule1 ; rule0.
rewrite :- try, rewrite.
rewrite.
```

similar idea: *graph databases*

Ingredient for Quantum Supremacy

method *state vector* simulation with efficient representation!

1Q-ops $|a|^2 + |b|^2 = 1 = |c|^2 + |d|^2 : \begin{pmatrix} a & b \\ -\bar{b} & \bar{a} \end{pmatrix} \circ \begin{pmatrix} c & d \\ -\bar{d} & \bar{c} \end{pmatrix} =$

$$\begin{pmatrix} ac - b\bar{d} & ad + b\bar{c} \\ -(\bar{a}\bar{d} + \bar{b}c) & \bar{a}\bar{c} - \bar{b}d \end{pmatrix} = \begin{pmatrix} ac - b\bar{d} & ad + b\bar{c} \\ -(\overline{ad} + \overline{b\bar{c}}) & \overline{ac} - \overline{b\bar{d}} \end{pmatrix}$$

2Q-ops

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \circ \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} a \\ b \\ d \\ c \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \circ \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \\ -d \end{pmatrix}$$

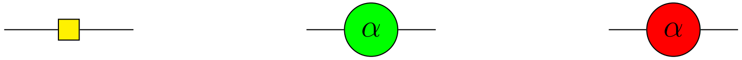
nQ-ops SWAP in context $|A\ 0\ B\ 1\ C\rangle \leftrightarrow |A\ 1\ B\ 0\ C\rangle$
Toffoli $|A\ 1\ B\ 1\ C\ 0\ D\rangle \leftrightarrow |A\ 1\ B\ 1\ C\ 1\ D\rangle$
Fredkin $|A\ 1\ B\ 0\ C\ 1\ D\rangle \leftrightarrow |A\ 1\ B\ 1\ C\ 0\ D\rangle$

Forschungsinstitut Cyber Defence
Universität der Bundeswehr München

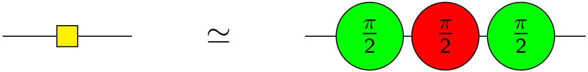
High-Performant, Concurrent, Parallel, Distributed

design	for high performance, parallel/distributed computation
efficient native code	nearly as fast as C, JIT via LLVM
coroutines	lightweight "green" threading
parallelism	w/(o) MPI, OpenMP-style • asynchronous: tasks, channels, events • multi-threading: atomic operations • distributed computing: parallel map+loops, RemoteChannels, SharedArrays
GPUs	e.g. CUDA for Nvidia library, OpenCL
interaction	REPL, Jupyter
distribution	core language, libraries, packages

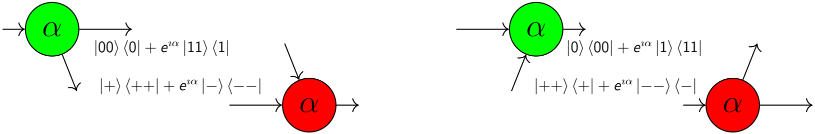
ZX-Calculus (with sound+complete rule set!)

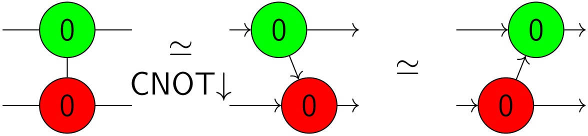


$$|+\rangle\langle 0| + |-\rangle\langle 1| \qquad |0\rangle\langle 0| + e^{\imath\alpha}|1\rangle\langle 1| \qquad |+\rangle\langle +| + e^{\imath\alpha}|-\rangle\langle -|$$



$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \simeq \begin{pmatrix}1 & 0\\ 0 & \imath\end{pmatrix}\begin{pmatrix}\frac{\imath}{1+\imath} & \frac{1}{1+\imath}\\ \frac{1}{1+\imath} & \frac{\imath}{1+\imath}\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & \imath\end{pmatrix} = \begin{pmatrix}\frac{\imath}{1+\imath} & \frac{\imath}{1+\imath}\\ \frac{\imath}{1+\imath} & \frac{-\imath}{1+\imath}\end{pmatrix} = \frac{\imath}{1+\imath}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$$





from square to rectangular matrices