Quantum-accelerated scientific computing

Matthias Möller

Delft Institute of Applied Mathematics

About me

- Assistant Professor in Numerical Analysis at TU Delft since 2013 (before TU Dortmund)

- Research interests:
  - FEM/IGA for (compressible) flow problems
  - High-resolution and high-order methods
  - Efficient multilevel solution methods
  - Hybrid particle mesh methods (MPM, OTM)
  - Heterogeneous high-performance computing
  - Quantum-accelerated scientific computing

TU Delft

State-of-the art in quantum-accelerated scientific computing and how we try to advance it

TU Delft

Accelerated computing

- Heterogeneous compute nodes with multi-socket, multi-core CPUs and general-purpose accelerators (GPUs, FPGAs, vector processors, …)
- **Current and future trend**: special-purpose accelerators (Google's TPUs, ASICs, …)
- **Vision**: use QPUs as functional accelerators
- **Philosophy**: Hardware-oriented Numerics = co-design of hardware-aware numerical methods and their hardware-optimized H^2PC implementation

TU Delft

QPUs

- **Discrete gate model**:
  - Google “Bristlecone”: 72(?) hw-qubits
  - IBM Q Experience: 4-16/20(?) hw-qubits
  - Intel ”Tangle Lake”: 49(?) hw-qubits
  - Rigetti: 19/128(?) hw-qubits
  - Atos QLM: 40 sw-qubits
  - QuTech QX/OpenQL: 26 sw-qubits
  - TNO Quantum Inspire: 31-37 sw-qubits

20 hw
40 sw
qubits

- ~~**Quantum annealing**:~~
  - ~~D-Wave system 2000Q: 2048/5000(?) qubits~~

TUDelft

Quantum SDKs

- **Quantum Assembly/Instruction languages**:
  - AQASM: Atos QML
  - cQASM: TNO Quantum Inspire, QuTech QX
  - OpenQASM: IBM Q Experience, Google
  - Quil: Rigetti simulator and cloud platform

- **SDKs** (in Python):
  - pyAqasm: Atos (AQASM in/out)
  - pyQuil: Rigetti (Quil in/out)
  - Circ: Google (OpenQASM out, no in)
  - QX/OpenQL (C++): QuTech (cQASM in/out)
  - ProjectQ: ETHZ (no xQASM in/out)
  - QisKit: IBM (OpenQASM in/output)
  - Quantum Development Kit (Q#): Microsoft (OpenQASM in/out)

TU Delft

Proprietary workflow

- Python script → [xQASM kernel] → QPU-optimized binary code → QSim/QComputer → post-processing in Python
- **Pros:**
  - Exploitation of knowledge of QPU internals in optimization
  - Flat learning curve to get started with basic quantum algorithm
- **Cons**:
  - Proprietary Q-toolchains (compilers, optimizers) and workflows
  - Re-inventing the wheel in each SDK (only prototype circuits)
  - No direct comparison of algorithms between QPUs possible
  - None of the tools aims at scientific computing at large scale
  - Investment insecurity (NVIDIA CUDA vs. ATI Stream SDK)

LibKet

|LIB⟩

- Kwantum expression template Library
- Header-only C++14 open-source library (soon) available at https://gitlab.com/mmoelle1/LibKet
- **Planning:** 1st official release before this summer with full support for all aforemention Q-backends
- **Long-term vision**: LibKet becomes the Eigen library of the Q-accelerated scientific computing community

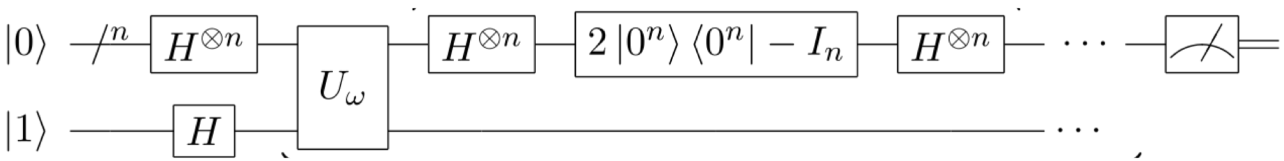
TUDelft

LibKet

|LIB⟩

- Provides C++ wrappers for all basic **quantum gates** and commonly used **circuits** templated over #qubits

```
auto expr = … h(all(x(sel<n>(init()))));
```



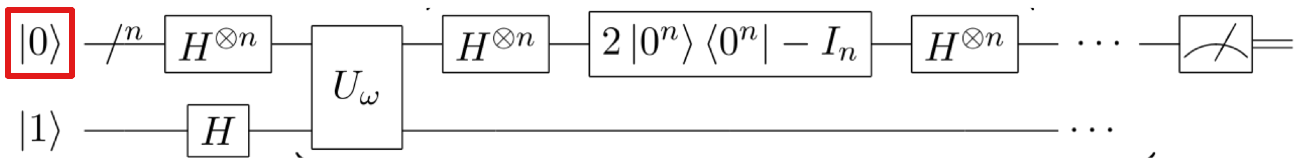
TU Delft

LibKet

|LIB⟩

- Provides wrappers for all basic **quantum gates** and commonly used **circuits** templated over #qubits

```
auto expr = … h(all(x(sel<n>(init()))));
```

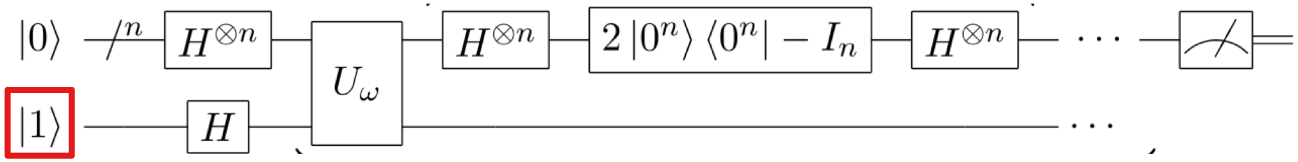


LibKet

$|\text{LIB}\rangle$

- Provides wrappers for all basic **quantum gates** and commonly used **circuits** templated over #qubits

```
auto expr = … h(all(x(sel<n>(init()))));
```

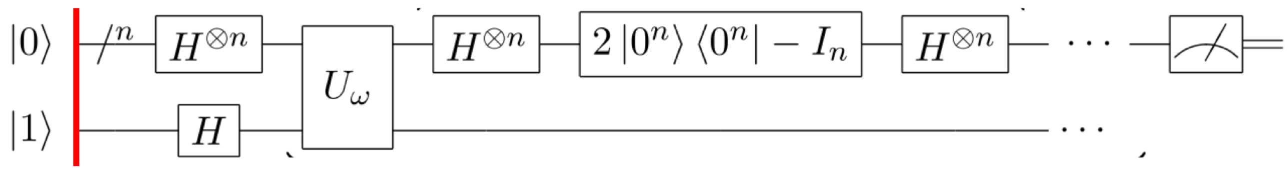


LibKet

|LIB⟩

- Provides wrappers for all basic **quantum gates** and commonly used **circuits** templated over #qubits

```
auto expr = … h(all(x(sel<n>(init()))));
```

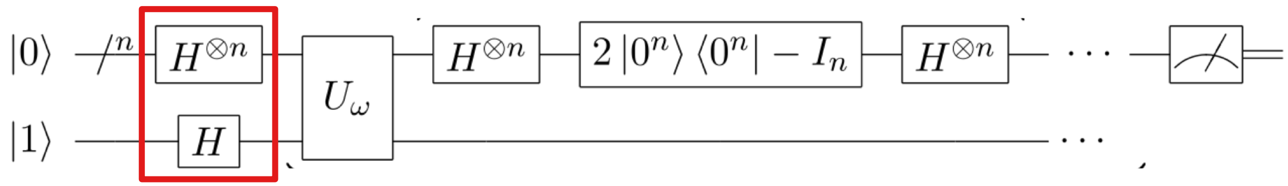


LibKet

|LIB⟩

- Provides wrappers for all basic **quantum gates** and commonly used **circuits** templated over #qubits

```
auto expr = … h(all(x(sel<n>(init()))));
```



LibKet

|LIB⟩

- Synthesizes quantum expressions into **rule-based optimized xQASM (QX/OpenQL) quantum kernels**

```
# cQASM 1.0
version 1.0
qubits 6
x q[5]
h q[0,1,2,3,4,5]
…
```

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[6];
x q[5];
h q[0];
h q[1];
h q[2];
…
```

TUDelft

LibKet

|LIB⟩

- Bidirectional communication between C++ host code and Python-based QSim/QComputer environment

```
QData<6, OpenQASMv2> backend;
json result = expr(backend).execute();
cout << result << endl;
```


```
[{"data":{"counts":{"0x0":22,"0x1":15,"0x10":18
,"0x11":11,"0x12":15,"0x13":11,"0x14":14,"0x15"
:20,"0x16":12,"0x17":13,"0x18":15,"0x19":16 ...
```


TU Delft

LibKet

|LIB⟩

- Will allow end-user to **develop quantum algorithms** from scratch but also to exploit Q-acceleration using ready-to-use **pre-built quantum expressions**

```
auto expr = qft<…>(range<0,5>(init()));
```

- Will provide **intrinsic types and arithmetic ops**:

```
QInt<6>      a(1),   b(1);   a+=b;
QPosit<8,1> a(1.3), b(2.3); a+=b;
```

TU Delft

LibKet workflow

|LIB⟩

- C++ host code
  → auto-generate rule-based optimized xQASM kernel
  → apply proprietary toolchain (compile & execute)
  → import results into C++ host code via JSON objects

- **Pros**:
  - Develop Q-accelerated scientific application in C++ only
  - Develop backend-independent quantum algorithms (QA) just once
  - Exploit all benefits (QPU-optimization) from proprietary toolchains
  - Compare different QPUs at the cost of a single code compilation

- **Cons**: None? Try it yourself and please tell me if any!

TU Delft

Past and ongoing activities

- **Bachelor projects**:
  - v.d. Lans: Multi-search Groover, Q-add/sub
  - Looman: Q-add with simulated quantum errors QC1.0
  - v.d. Linde: Posit arithmetics

  - Driebergen: Posit arithmetics for QC
  - Ubbes: Quantum Linear Solver Algorithm (QLSA) QC2.0
  - Schalkers (internship at TNO): LibKet, …

- **Collaborations**:
  - TNO, TU Delft Quantum & Computer Eng., SurfSara

TU Delft

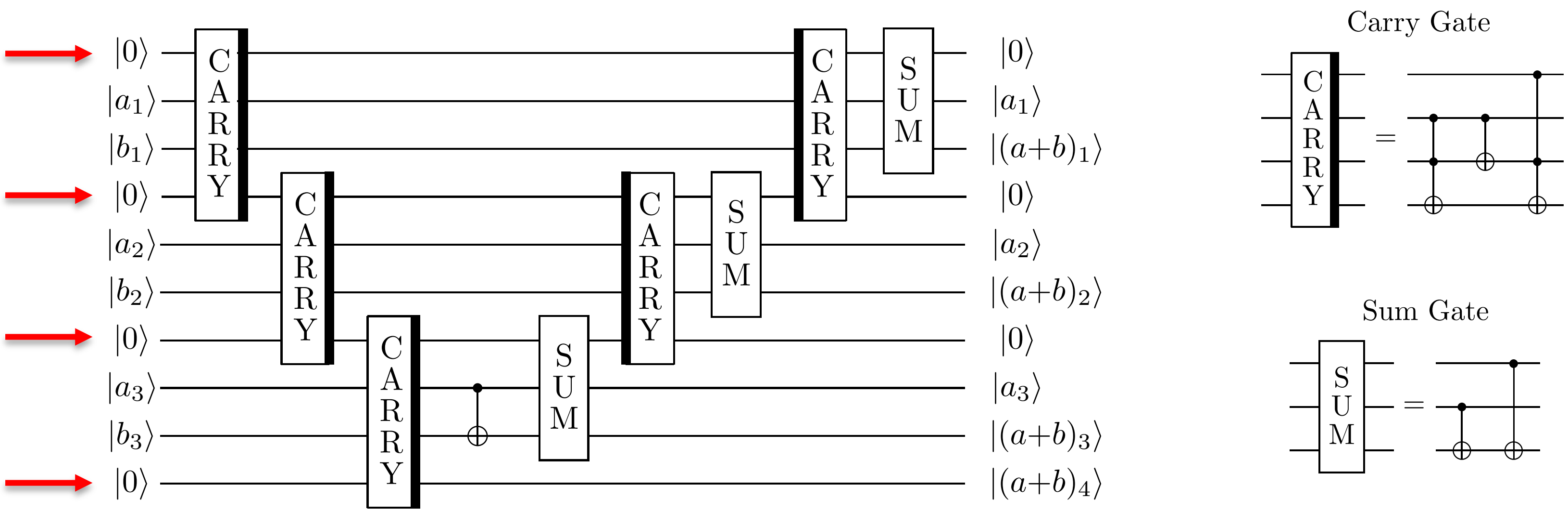
A first quantum algorithm: 1+1=2

- **Integer addition**:
  - a = 1; b = 1; c = a+b; ✗ (no-cloning principle)
  - a = 1; b = 1; a += b; ✓

- **Classical adder circuits**:
  - Must be reversible (all QAs are reversible!)
  - Must be realizable with quantum gates only
  - Should need few ancilla qubits (20-40 qubits)

TU Delft

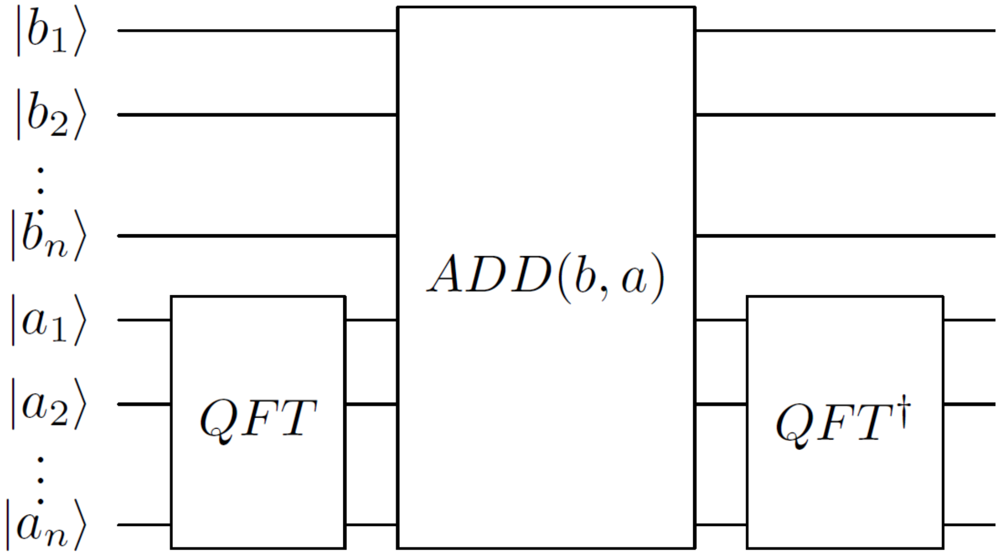
A first quantum algorithm: 1+1=2



n extra ancilla qubits needed ☹

Cuccaro et al.: A new quantum ripple-carry addition circuit (2008)

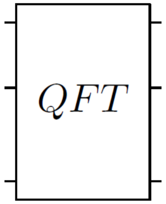
Another quantum algorithm: 1+1=2

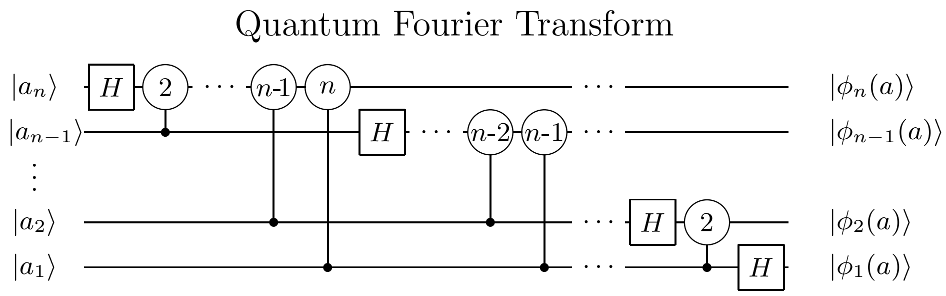


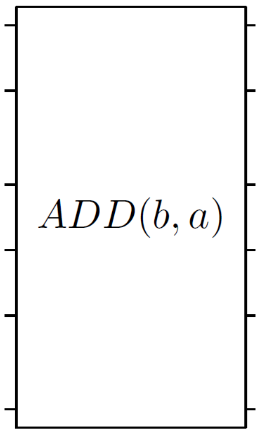
no extra ancilla qubits needed ☺

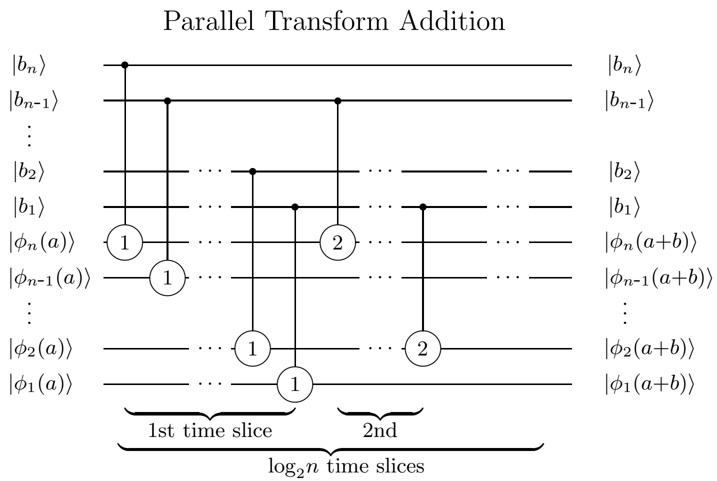
Draper: Addition on a quantum computer (2000)

Another quantum algorithm: 1+1=2









Draper: Addition on a quantum computer (2000)

Using LibKet: 1+1=2

- **LibKet quantum expression**:

```
auto expr =
  qftdag( range<n,n>( add( range<0,n>( all() ),
                           qft( range<n,n>() )
        )           );
```

- Or simply (planned for 1st official release):

```
QInt<6> a(1), b(1); a += b;
```

TUDelft

Towards practical QC: 1+1 ≅ 2

1000 QX simulator runs with depolarizing noise error model

QInt<n>	0,1		$10^{-\frac{3}{2}}$		0,01		$10^{-\frac{5}{2}}$	
1	0.27045	0.3793	0.50545	0.2752	0.78965	0.1233	0.92285	0.0463
2	0.134061	0.221523	0.165182	0.209176	0.451353	0.134284	0.762621	0.0570876
3	0.0601436	0.112097	0.0683512	0.116162	0.191802	0.105916	0.540766	0.0754021
4	0.0336509	0.0611537	0.0351125	0.0589036	0.064375	0.0645881	0.306778	0.0802711
5					0.0224336	0.031892	0.154869	0.0575671
6					0.00798384	0.0176539	0.0654961	0.033179
7					0.00398747	0.0076473	0.0252142	0.0167067
8					0.00254026	0.00363275	0.00834128	0.00823629

Standard circuit: prob. correct (left), largest prob. wrong answer (right)

Looman: Implementation and Analysis of an Algorithm on Positive Integer Addition for Quantum Computing (2018)

Towards practical QC: 1+1 ≅ 2

1000 QX simulator runs with depolarizing noise error model

QInt<n>	0,1		$10^{-\frac{3}{2}}$		0,01		$10^{-\frac{5}{2}}$	
1	0.29475	0.3695	0.54555	0.27185	0.8158	0.11735	0.93645	0.04195
2	0.110416	0.230068	0.239152	0.203304	0.569495	0.115691	0.837026	0.0445888
3	0.0581316	0.114572	0.096711	0.122477	0.341537	0.102147	0.697436	0.0509187
4	0.0259028	0.0583002	0.0382769	0.0672328	0.183066	0.0726129	0.543162	0.0579935
5					0.0839273	0.0450361	0.407117	0.0574072
6					0.0412412	0.0270095	0.283642	0.049151
7					0.0177059	0.0131818	0.191996	0.0404665
8					0.00647699	0.00675828	0.116269	0.0290022

Optimized circuit: prob. correct (left), largest prob. wrong answer (right)

Looman: Implementation and Analysis of an Algorithm on Positive Integer Addition for Quantum Computing (2018)

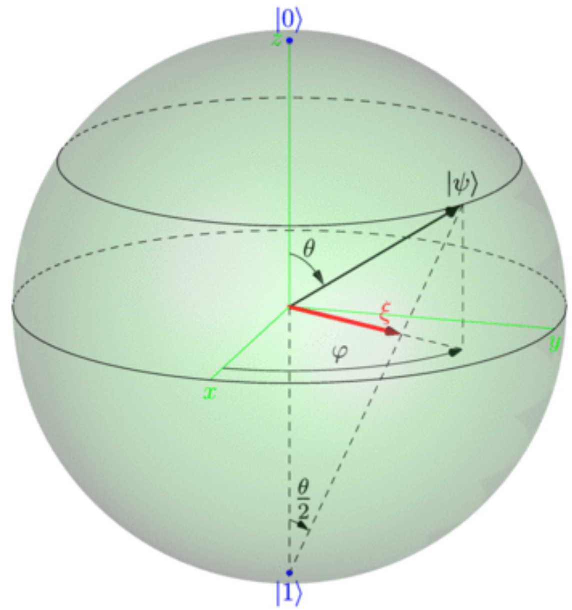
Quantum computing in a nutshell and why it's so difficult to make progress

QC in a nutshell

- **A single qubit state:**

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle,$$
$$\alpha, \beta \in \mathbb{C}, |\alpha|^2 + |\beta|^2 = 1$$

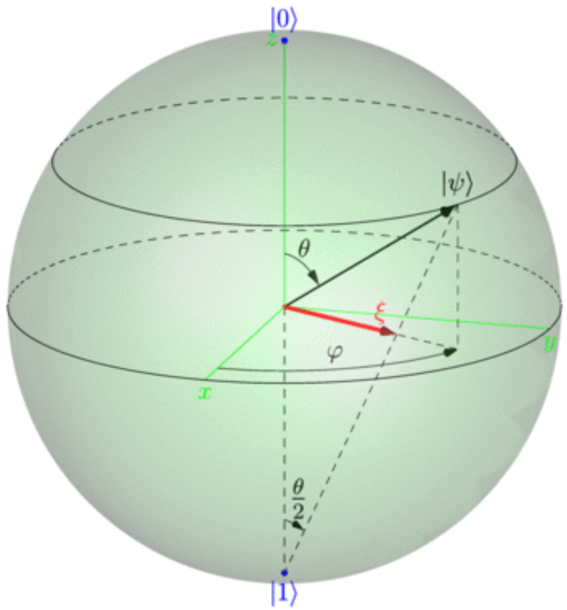
$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle$$



QC in a nutshell

- **Hadamard (H) gate:**

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

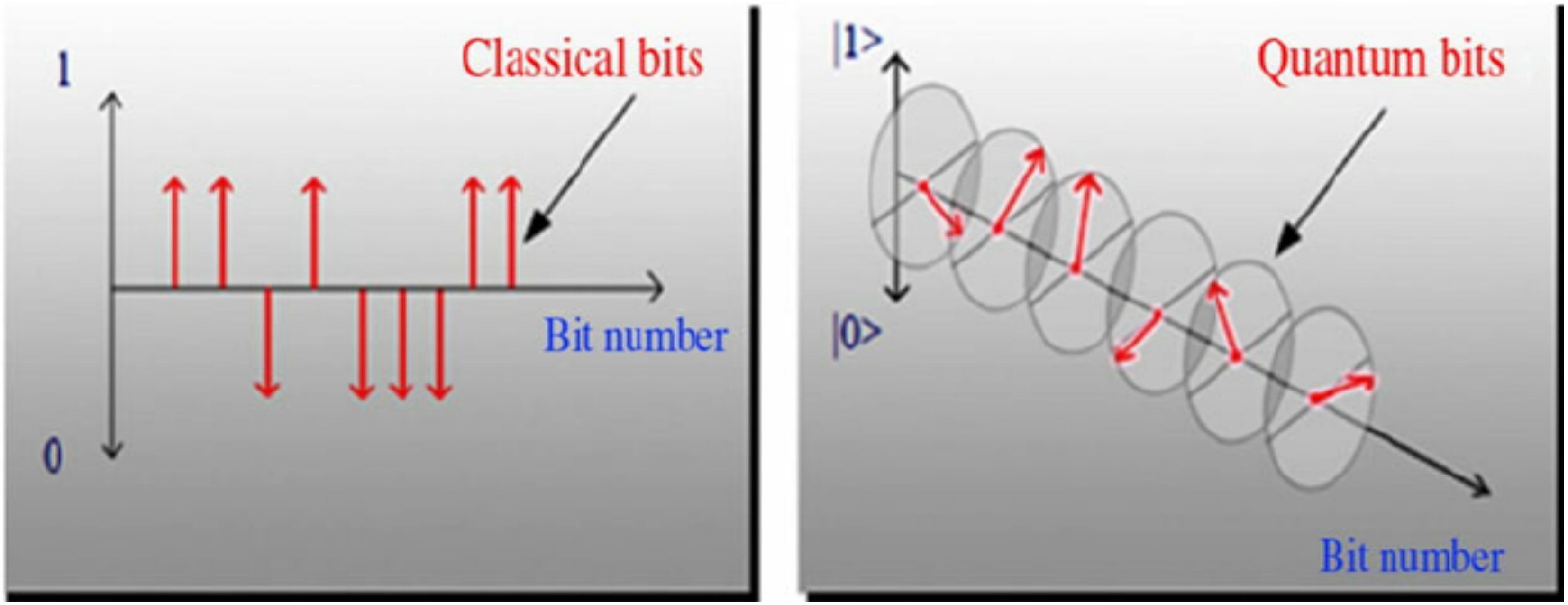


Unitary operators, i.e. $UU^{\dagger} = I$ or $\langle u, v \rangle_H = \langle Uu, Uv \rangle_H, \forall u, v \in H$

$$H|\psi\rangle = \frac{\alpha + \beta}{\sqrt{2}} |0\rangle + \frac{\alpha - \beta}{\sqrt{2}} |1\rangle$$

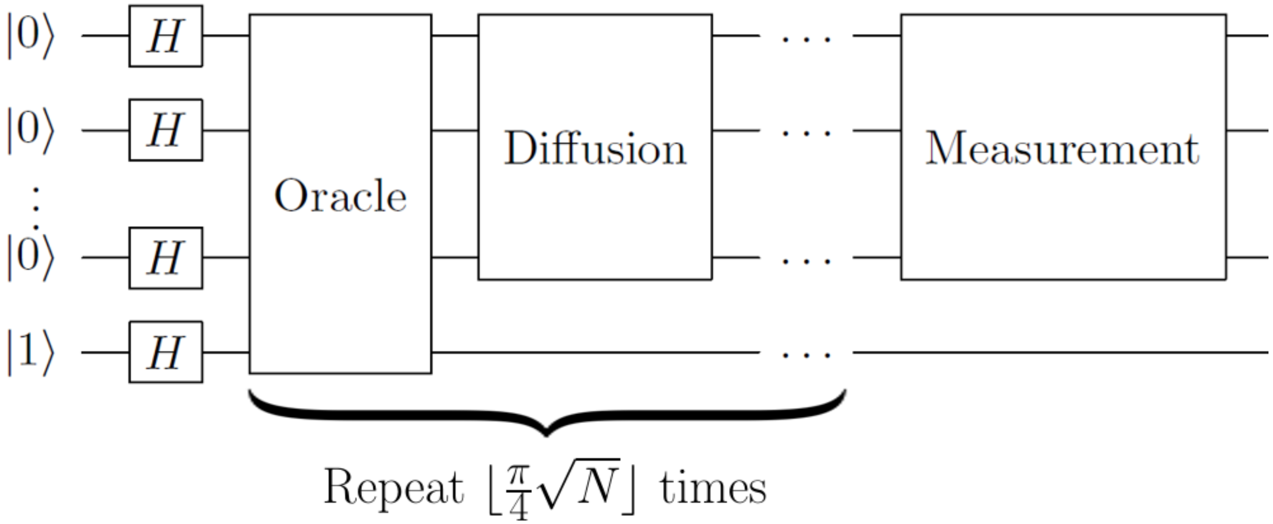
TU Delft

QC in a nutshell



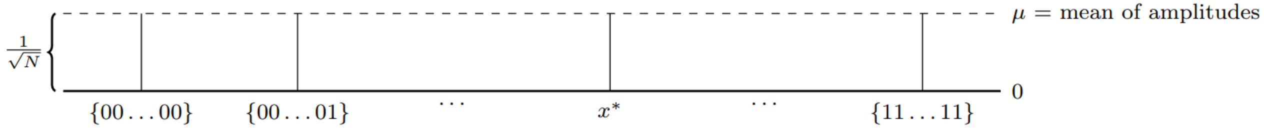
Farouk et al.: Architecture of multicast centralized key management scheme using Quantum key distribution and classical symmetric encryption (2014)

Grover's search algorithm



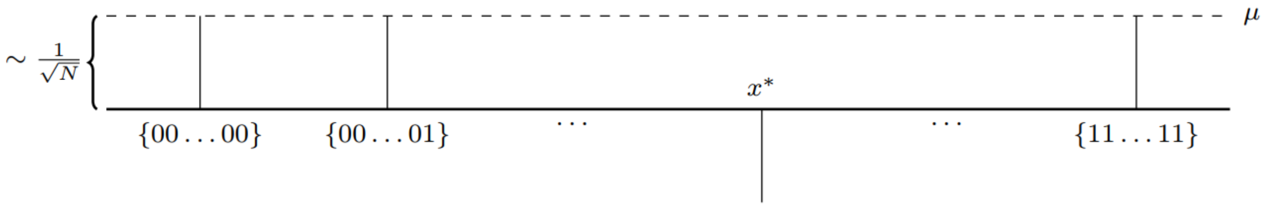
v.d. Lans: QAs and their implementation on QC simulators (2017)

Grover's search algorithm



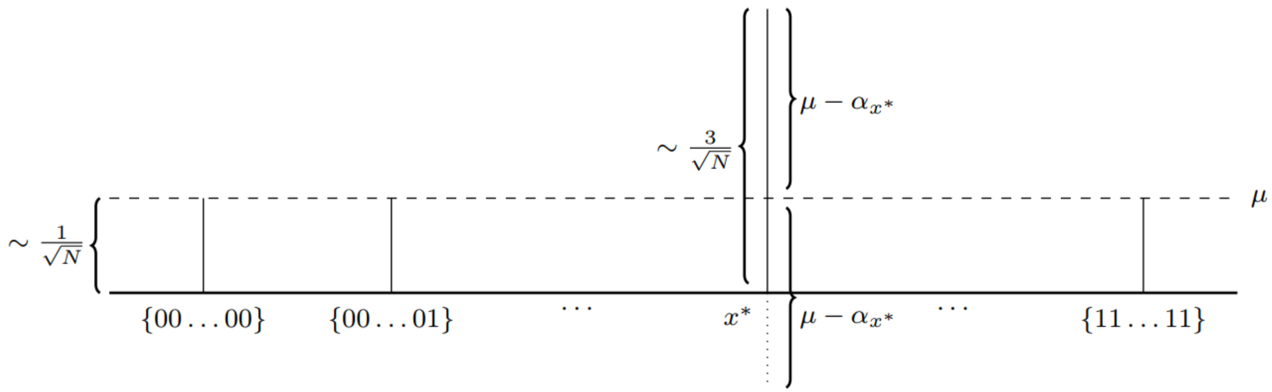
Wright, Tseng: Grover's algorithm (2015)

Grover's search algorithm



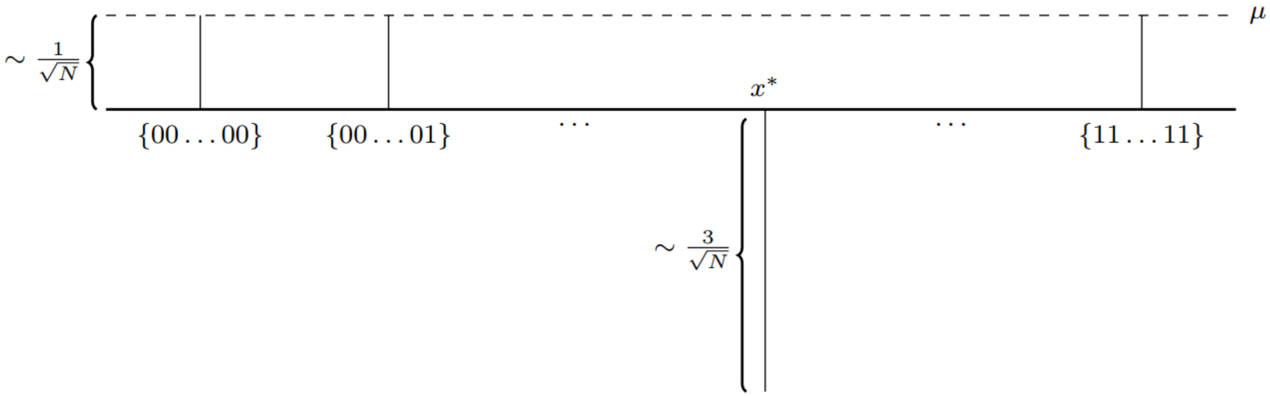
Wright, Tseng: Grover's algorithm (2015)

Grover's search algorithm



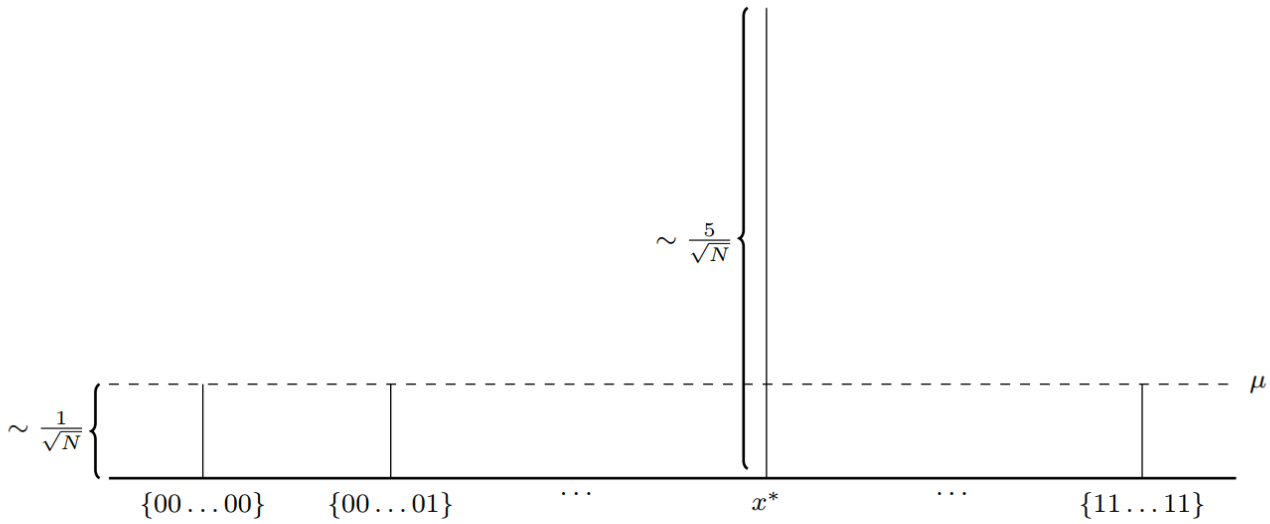
Wright, Tseng: Grover's algorithm (2015)

Grover's search algorithm



Wright, Tseng: Grover's algorithm (2015)

Grover’s search algorithm



Wright, Tseng: Grover’s algorithm (2015)

Long-term vision

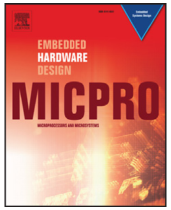
Microprocessors and Microsystems 66 (2019) 67–71

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

ELSEVIER



A conceptual framework for quantum accelerated automated design optimization

Check for updates

Matthias Möller[a,*], Cornelis Vuik[a]

Delft University of Technology, Delft Institute of Applied Mathematics (DIAM), Van Mourik Broekmanweg 6, XE Delft 2628, The Netherlands

TU Delft

Design optimization

- **Abstract problem**:

$$\min_{\boldsymbol{\alpha}\in\mathcal{D}} \mathcal{J}(U(\boldsymbol{\alpha}); Y) \quad \text{s.t.} \quad \mathcal{R}(U(\boldsymbol{\alpha}); Y) = 0$$

  - Admissible design parameters $\boldsymbol{\alpha} \in \mathcal{D}$
  - Generated design (control) $U(\boldsymbol{\alpha})$
  - Solution $Y = Y(U(\boldsymbol{\alpha}))$ to PDE in residual form
  - Cost functional $\mathcal{J}(\cdot)$ to be minimized

TU Delft

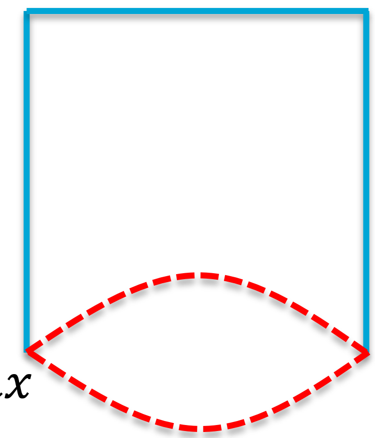
Academic model problem

- **2D Poisson equation**:

$$
\begin{aligned}
-\Delta u &= f \text{ in } \Omega(\alpha) \\
u &= 0 \text{ on } \Gamma(\alpha)
\end{aligned}
$$

- **Design parameter**:

$$y(x) = \alpha(x - x^2), \alpha^{min} \leq \alpha \leq \alpha^{max}$$



- **Optimization problem**:
  - Minimize L_2-error between solution u and a given reference profile u^* by adjusting the shape of the domain boundary at the bottom

TU Delft

Q-accelerated linear solvers

- **Discretized problem**:

$$\min_{\alpha \in \mathcal{D}} \left(\int_{\Omega(\alpha)} y_h^2 \, dx \right)^{\frac{1}{2}} \approx \min_{\alpha \in \mathcal{D}} \left(y_h^T M y_h \right)^{\frac{1}{2}}$$

sparse matrix

such that

s-sparse SPD well-conditioned matrix

$$A_h y_h = f_h - A_h u_h^*$$
$$y_h = u_h - u_h^*$$

TUDelft

Q-accelerated optimization

- **Taylor expansion about the optimal state $\boldsymbol{\alpha}^*$:**

$$\mathcal{J}\left(\boldsymbol{\alpha}^{(k)}\right) - \mathcal{J}(\boldsymbol{\alpha}^*) =$$

$$\cancel{\sum_{i=1}^{\dim \mathcal{D}} \left.\frac{\partial \mathcal{J}}{\partial \alpha_i}\right|_{\boldsymbol{\alpha}^*} \left(\alpha_i^{(k)} - \alpha_i^*\right)}$$

$$+\frac{1}{2}\sum_{i,j=1}^{\dim \mathcal{D}} \left(\alpha_i^{(k)} - \alpha_i^*\right) \left.\frac{\partial^2 \mathcal{J}}{\partial \alpha_i \partial \alpha_j}\right|_{\boldsymbol{\alpha}^*} \left(\alpha_i^{(k)} - \alpha_i^*\right)$$

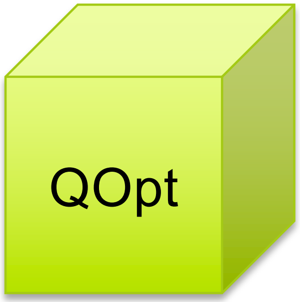
$$+\mathcal{O}\left(\left\|\boldsymbol{\alpha}^{(k)} - \boldsymbol{\alpha}^*\right\|^3\right)$$

TU Delft

Q-accelerated optimization

- **Positive-definite quadratic form**:

$$\mathcal{Q}\left(\boldsymbol{\alpha}^{(k)}\right) = \frac{1}{2} \sum_{i,j=1}^{\dim \mathcal{D}} \left(\alpha_i^{(k)} - \alpha_i^*\right) \frac{\partial^2 \mathcal{J}}{\partial \alpha_i \partial \alpha_j}\bigg|_{\boldsymbol{\alpha}^*} \left(\alpha_i^{(k)} - \alpha_i^*\right)$$



Potential Q-speedup

- **QLSA**:
  - CG method: $\mathcal{O}(N s\kappa \log(1/\epsilon))$
  - HHL (2009): $\mathcal{O}(\log(N) s^2\kappa^2/\epsilon)$
  - Ambainis (2012): $\mathcal{O}(\log(N) s^2\kappa/\epsilon)$
  - Childs et al. (2017): $\mathcal{O}(\text{polylog}(s\kappa/\epsilon) s\kappa)$

- **QOpt**:
  - Yao (1975): $\mathcal{O}(\dim \mathcal{D}^2)$
  - Jordan (2008): $\mathcal{O}(\dim \mathcal{D})$

- **Other**:
  - Cao et al. (2013): FDM Q-Poisson solver
  - Montanaro et al. (2016): FEM Q-Poisson solver

TU Delft


AIRBUS
Commercial Aircraft
Helicopters
Defence
Space
Company
Newsroom
Airbus Home › Innovation › Tech Challenges & Competitions › Airbus Quantum Computing Challenge
Aircraft Climb Optimization
Wingbox Design Optimization
CFD on Quantum Computers
Surrogate modelling of PDEs
Aircraft Loading Optimization
Airbus Quantum Computing Challenge
Bringing flight physics into the Quantum Era

Wrap-up

- **LibKet:**
  - Early adopter usage and feedback highly appreciated
- **Q-accelerated shape optimization**:
  - Feedback on concept and collaboration welcome
- **Possible collaboration with NLR**:
  - Airbus Quantum Challenge and other topics
  - Q-Flagship project (coordinated by K. Bertels)

Thank you for your attention!

TU Delft