Quantum-accelerated scientific computing: concepts, programming tools and applications

Matthias Möller

Delft University of Technology
Delft Institute of Applied Mathematics

Joint work with Boyang Chen, Marc Gerritsma, Zaid Al-Ars, Jens de Zoete, Giorgio Balducci, Enrico Cappanera, Smaran Adarsh, Merel Schalkers, Koen Mesman, Tim Driebergen, …
Quantum computing in the news

Quantum supremacy using a programmable superconducting processor

Recent advances in quantum computing have resulted in two 53-qubit processors: one from our group in IBM and a device described by Google in a paper published in the journal Nature. In the paper, it is argued that their device reached "quantum supremacy" and that "a state-of-the-art supercomputer would require approximately 10,000 years to perform the equivalent task." We argue that an ideal simulation of the same task can be performed on a classical system in 2.5 days and with far greater fidelity. This is in fact a conservative, worst-case estimate, and we expect that with additional refinements the classical cost of the simulation can be further reduced.
Quantum computing in Europe

THE FIRST EUROPEAN ONLINE QUANTUM COMPUTER PLATFORM

4 May 2020 • 3 min reading time

Leading universities and quantum hubs from China to America and the Netherlands are working on the development of a usable quantum computer. Within QuTech, TNO is working on innovative quantum technology in collaboration with Delft University of Technology and with some success, because a new version of the ‘Quantum Inspire’ quantum computing platform was launched on 20 April 2020. It is, in fact, the first European quantum computer platform that is generally accessible online.

Quantum Delta NL awarded 615 million euro from Netherlands’ National Growth Fund to accelerate quantum technology
Quantum-accelerated scientific computing

- concepts
  - qubits, gates, and simple algorithms

- programming tools
  - *LibKet* and generation of resource-optimal quantum circuits

- applications
  - quantum linear solvers and optimization algorithms

- Conclusion
concepts

qubits, gates, and simple algorithms
int a = 1;
int b = 2;
int c = a+b;
int a = 1;
int b = 2;
int c = a + b;

ld r0 mem(a) 10001100000010100000110000100000
ld r1 mem(b) 10001100010010110000001001100010
add r0 r1 r2 10101101100010100000010110100110
sd r2 mem(c) 10000100100010100000010000011011
von Neumann model

- Input device
- Output device
- Central processing unit
  - Control unit
  - Arithmetic/logic unit
- Memory unit

Intel® Core™ i7-3960X Processor
a quantum computer model

- superconducting
- trapped ion
- quantum dots
- NV centers
- photonics (room temperature)

- 300 K ~ 27°C
- 4 K ~ -269°C
- 20 mK ~ -273°C
IBM’s 27-qubit processor

data is ‘stored’ in qubits and can be manipulated by 1- and 2-qubit ‘gates’

2-qubit gates between nonadjacent qubits require additional ‘swap’ ops

controlled-NOT gate between q3 and q9

swap q3 q5
swap q9 q8
cnot q5 q8
quantum bits

- **qubit**: quantum version of a bit

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

- computational **basis**

  \[ \mathcal{E} = (|0\rangle, |1\rangle) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \]

- coefficients \( \alpha, \beta \) are the **probability amplitudes** and \( |\alpha|^2 \) and \( |\beta|^2 \) are the **probabilities** of measuring the basis states \( |0\rangle \) and \( |1\rangle \), respectively
single-qubit states

- **Bloch sphere**

\[ |\psi\rangle = e^{i\delta} \left( \cos \frac{\theta}{2} |0\rangle + e^{i\varphi} \sin \frac{\theta}{2} |1\rangle \right) \]

- polar angle \( \theta \in [0, \pi] \)
- azimuthal angle \( \varphi \in [0, 2\pi] \)
- global phase \( \delta \)
classical gates

- **NOT**
  
  ![NOT gate diagram]

<table>
<thead>
<tr>
<th>A</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- **NAND**

  ![NAND gate diagram]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- logical operations based on truth tables
- most classical gates are not reversible
quantum gates

- **Pauli X**

  ![Pauli X Gate]

  $$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

- **Hadamard**

  ![Hadamard Gate]

  $$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

- Unitary operations represented by unitary matrices
- All quantum gates are reversible, e.g. $HH^\dagger = I$
single-qubit gates

$|0\rangle$  
\[ \begin{array}{c|c} 
\hline 
& \bigg| \bigg. \\
X & 1 \\
& \bigg| \bigg. 
\hline 
\end{array} \bigg| 1 \bigg. 
\]

$|0\rangle$  
\[ |+\rangle := \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \]

$|0\rangle$  
\[ \begin{array}{c|c} 
\hline 
& \bigg| \bigg. \\
H & 1 \\
& \bigg| \bigg. 
\hline 
\end{array} \bigg| 1 \bigg. 
\]
single-qubit gates

\[ |0\rangle \xrightarrow{X} |1\rangle \quad |1\rangle \xrightarrow{X} |0\rangle \]

\[ |0\rangle \xrightarrow{H} \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \quad |1\rangle \xrightarrow{H} \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \]
**single-qubit circuits**

\[ |\psi_{in}\rangle \xrightarrow{U_1} U_1 |\psi_{in}\rangle \xrightarrow{U_2} U_2 U_1 |\psi_{in}\rangle \xrightarrow{U_3} U_3 U_2 U_1 |\psi_{in}\rangle = |\psi_{out}\rangle \]

- single-qubit gates $U_k$ are **unitary matrices**, i.e.
  \[ U_k U_k^\dagger = U_k^\dagger U_k = I \]

- quantum circuits are sequences of matrix-vector multiplications
  \[ |\psi_{out}\rangle = U_3 U_2 U_1 |\psi_{in}\rangle \]
multi-qubit states

- $|\psi_0\rangle = \alpha_0 |0\rangle + \beta_0 |1\rangle = \alpha_0 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta_0 \begin{pmatrix} 0 \\ 1 \end{pmatrix}$
- $|\psi_1\rangle = \alpha_1 |0\rangle + \beta_1 |1\rangle = \alpha_1 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta_1 \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

| $\alpha_0 \alpha_1 |00\rangle + \alpha_0 \beta_1 |01\rangle + \beta_0 \alpha_1 |10\rangle + \beta_0 \beta_1 |11\rangle =: |\psi_0\psi_1\rangle$

with $|\alpha_0 \alpha_1|^2 + |\alpha_0 \beta_1|^2 + |\beta_0 \alpha_1|^2 + |\beta_0 \beta_1|^2 = 1$
multi-qubit states

- tensor product of $n$ single-qubit states

$$|\psi_0 \ldots \psi_n\rangle = \gamma_{0\ldots00}|0 \ldots 00\rangle + \gamma_{0\ldots01}|0 \ldots 01\rangle + \ldots + \gamma_{1\ldots11}|1 \ldots 11\rangle$$

- an $n$-qubit register can hold the $2^n$ inputs ‘simultaneously’ in superposition

- a few words of caution
  - it is impossible to obtain the $\gamma$’s; one obtains a single binary answer, say, $|001101\rangle$ with probability $|\gamma_{001101}|^2$ upon measurement

  - a single run of a quantum circuit is not very useful; many runs are required to measure the correct answer with sufficient certainty
Example: 3-bit password

<table>
<thead>
<tr>
<th>Classical:</th>
<th>Quantum:</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td></td>
</tr>
<tr>
<td>001</td>
<td></td>
</tr>
<tr>
<td>010</td>
<td></td>
</tr>
<tr>
<td>011</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td></td>
</tr>
</tbody>
</table>

Password: __________  →  010

Grover's algorithm
Grover’s algorithm

- Quantum circuit on QI

- Quantum circuit on IBM
multi-qubit gates

\[ |\Psi_{\text{in}}\rangle \quad H \quad |\Psi_{\text{out}}\rangle = H \otimes I |\Psi_{\text{in}}\rangle \]

\[
H \otimes I |00\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix}
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & -1 & 0 \\
0 & 1 & 0 & -1
\end{pmatrix} \begin{pmatrix}
1 \\
0 \\
0 \\
0
\end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix}
1 \\
0 \\
0 \\
0
\end{pmatrix} = \frac{|00\rangle + |10\rangle}{\sqrt{2}} = \frac{(|0\rangle + |1\rangle) \otimes |0\rangle}{\sqrt{2}}
\]
entanglement

$CNOT(H \otimes I)|00\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$

- Bell state is maximally entangled. By measuring one of the two qubits one knows the value of the other qubit without a further measurement.
quantum teleportation

\[ \alpha|0\rangle + \beta|1\rangle \]

Share EPR Pair

Encode

Measure

Send

Decode

Received

\[ \alpha|0\rangle + \beta|1\rangle \]
quantum teleportation

\[ |010\rangle - |110\rangle + |001\rangle - |101\rangle \]

\[ \frac{2}{2} \]
quantum teleportation

\[ |010\rangle - |110\rangle + \frac{|001\rangle - |101\rangle}{2} \]
quantum teleportation

\[ |010\rangle - |110\rangle + |001\rangle - |101\rangle \]

\[ \frac{2}{2} \]
quantum teleportation

\[ |010\rangle - |110\rangle + |001\rangle - |101\rangle \]

\[ \frac{2}{2} \]

|q_0\rangle \quad X \quad 1

|q_1\rangle \quad H

|q_2\rangle

H

H

1

0

-1

X

Z

1
quantum teleportation

\[ |010⟩ - |110⟩ + |001⟩ - |101⟩ \]

\[
\frac{1}{2} \]

\[
|q_0⟩ \quad \text{X} \quad 1
\]

\[
|q_1⟩ \quad \text{H} \quad 1
\]

\[
|q_2⟩ \quad \text{H} \quad 1 \quad -0 \quad \text{X} \quad \text{Z} \quad 1
\]
programming tools

LibKet and generation of resource-optimal quantum circuits
Lib – The kquantum expression template Library

- IBM Q
- Rigetti
- Xanadu
- Honeywell
- IonQ
- D-Wave
- AQFT
- Quantum Inspire

Companies providing quantum computing services, with QuantumLib as a framework for quantum expression template library.
quantum acceleration workflow

C/C++

```c
q = expr(...)
while (cond)
{
    h = q.exec(...)
    // classical compute
    h.wait()
    ...
}
```
quantum acceleration workflow

C/C++

```c
q = expr(...)

while (cond)
{
    h = q.exec(...)
    // classical compute
    h.wait()
    ...
}
```

you  vendor
quantum acceleration workflow

C/C++

q = expr(...)

while (cond)
{
    h = q.exec(...)
    // classical compute
    h.wait()
    ...
}
different programming philosophies

standard quantum SDKs
- apply gates to individual qubits
  
  \[
  \begin{align*}
  &H_{q[0:2]} \\
  &X_{q[0,2]} \\
  &H_{q[2]} \\
  &CCX\ q[0],\ q[1],\ q[2] \\
  \end{align*}
  \]

LibKet
- ‘stream’ qubits through gates
  
  \[
  \begin{align*}
  &...CCX(q[0],\ q[1],\ q[2]) \\
  &\quad H(q[2])(\ X(q[0,2])(\ H(q[0:2]()) \\
  &\quad \quad )) \\
  &\quad )) \\
  &))
  \end{align*}
  \]
different programming philosophies

standard quantum SDKs

- apply gates to individual qubits

LibKet

- ‘stream’ qubits through gates

```
H q[4, 7, 8]
X q[4, 8]
H q[8]
CCX q[4], q[7], q[8]
...
```

```
...CCX(q[0],
    q[1],
    q[2](
        H(q[2](
            X(q[0, 2](
                H(q[0:2](q[4, 7, 8]))
            ))
        ))
    ))
```
filters

- selective ‘views’ on the qubits

```cpp
auto f0 = select<0,2,3>();
```
filters

- selective ‘views’ on the qubits

```cpp
auto f0 = select<0,2,3>();
auto f1 = range<1,2>(f0);
```
filters

- selective ‘views’ on the qubits

```cpp
auto f0 = select<0,2,3>();
auto f1 = range<1,2>(f0);
auto f2 = tag<0>(f1);
```
filters

- selective ‘views’ on the qubits

```cpp
auto f0 = select<0,2,3>();
auto f1 = range<1,2>(f0);
auto f2 = tag<0>(f1);
auto f3 = qubit<1>(f2);
```
filters

- selective ‘views’ on the qubits

```cpp
auto f0 = select<0,2,3>();
auto f1 = range<1,2>(f0);
auto f2 = tag<0>(f1);
auto f3 = qubit<1>(f2);
auto f4 = tag<1>(f3);
```
filters

- selective ‘views’ on the qubits

```cpp
auto f0 = select<0,2,3>();
auto f1 = range<1,2>(f0);
auto f2 = tag<0>(f1);
auto f3 = qubit<1>(f2);
auto f4 = tag<1>(f3);
auto f5 = gototag<0>(f4);
```
filters

- selective ‘views’ on the qubits

```cpp
auto f0 = select<0,2,3>();
auto f1 = range<1,2>(f0);
auto f2 = tag<0>(f1);
auto f3 = qubit<1>(f2);
auto f4 = tag<1>(f3);
auto f5 = gototag<0>(f4);
auto f6 = gototag<1>(f5);
```
gates

- SIMD-like quantum operation on all qubits of the current filter chain

```cpp
auto eθ = init();
```
gates

- SIMD-like quantum operation on all qubits of the current filter chain

```cpp
auto e0 = init();
auto e1 = sel<0,2>(e0);
```
- SIMD-like quantum operation on all qubits of the current filter chain

```cpp
gates

auto e0 = init();
auto e1 = sel<0,2>(e0);
auto e2 = h(e1);
```
SIMD-like quantum operation on all qubits of the current filter chain

```cpp
auto e0 = init();
auto e1 = sel<0,2>(e0);
auto e2 = h(e1);
auto e3 = all(e2);
```
gates

- SIMD-like quantum operation on all qubits of the current filter chain

auto e0 = init();
auto e1 = sel<0,2>(e0);
auto e2 = h(e1);
auto e3 = all(e2);
auto e4 = cnot(
    sel<0,2>(),
    sel<1,4>(e3)
);

```
  q_0  H  
  |     |
  q_1  X  
  |     |
  q_2  H  
  |     |
  q_3  X  
  |     |
  q_4  
```
gates

- SIMD-like quantum operation on all qubits of the current filter chain

```cpp
auto e0 = init();
auto e1 = sel<0, 2>(e0);
auto e2 = h(e1);
auto e3 = all(e2);
auto e4 = cnot(
    sel<0, 2>(e0),
    sel<1, 4>(e3)
);
auto e5 = measure(all(e4));
```
3-qubit Grover’s algorithm

```cpp
auto oracle = [] (auto expr) {
    return x (sel_<0> (x (h (sel_<2> (ccnot (sel_<0> ()) ,
                             sel_<1> () ,
                             sel_<2> (h (x (sel_<2> (x (sel_<0> (expr)))))))))));
};

auto diffusion = [] (auto expr) {
    return h (x (all (h (sel_<2> (ccnot (sel_<0> ()) ,
                              sel_<1> () ,
                              sel_<2> (h (x (sel_<2> (x (h (all (expr)))))))))));
};

auto expr = measure (diffusion (oracle (h (init ()))))
QDevice<backend, 3> device;
utils::json res = device (expr).eval (shots);
cout << device.get<QResultType::best> (res) << endl;
```
3-qubit Grover’s algorithm

- IBM’s basis gates: CX, ID, RZ, SX, X
- *executable* quantum circuit generated by IBM’s quantum compiler

```cpp
auto expr = measure(diffusion(oracles(h(init()))));
QDevice<QDeviceType::ibmq_quito, 3> device;
utils::json res = device(expr).eval(shots);
cout << device.get<QResultType::best>(res) << endl;
```
traditional quantum circuit compilation

- gate substitution rules

\[ H \rightarrow R_x(\pi)R_y(\pi/2), \quad H \rightarrow R_y(-\pi/2)R_x(\pi), \quad \ldots \]

- cancelling of inverse gates

\[ CZ \; CZ^\dagger = I, \quad R_x(\theta)R_x(-\theta) = I, \quad \ldots \]

- aggregation using commutativity or fusion rules

\[ HR_z(\theta)H = R_x(\theta), \; \theta \in \{\pi, \pm \pi/2\}, \quad R_z(\theta_1)R_z(\theta_2) = R_z(\theta_1 + \theta_2), \quad \ldots \]
approximate computing

- our aim is to generate a resource-efficient directly executable circuit $U(\theta)$ that mimics the expectation-value behavior of the textbook circuit $V$
approximate computing

- our aim is to generate a resource-efficient directly executable circuit $U(\theta)$ that mimics the expectation-value behavior of the textbook circuit $V$. 

![Diagram of quantum circuit](image-url)
approximate computing

$$U_{\text{opt}} = \arg\min_{U \in \mathcal{U}_s} \min_{\theta_U} \max_{|\psi\rangle \in \Psi} F(|\psi\rangle; V, U(\theta_U)), \quad s \to \min$$

- cost function

$$F(|\psi\rangle; V, U(\theta_U)) = \sum_k \left( \langle A^k \rangle_{V|\psi\rangle} - \langle A^k \rangle_{U(\theta_U)|\psi\rangle} \right)$$

- $A^k$ is an observable, e.g., Pauli-X, Y, Z gate
- expectation value of state $|\psi\rangle$ upon application of operator $P$

$$\langle A^k \rangle_{P|\psi\rangle} = \langle (P\psi)^\dagger |A^k |P\psi \rangle$$

- $\mathcal{U}_s$ is the set of all admissible quantum circuits of size $s$
- $U(\theta_U)$ is one parametrized quantum circuit with $\theta_U = (\theta_1, \ldots, \theta_N)^T$
<table>
<thead>
<tr>
<th>algo</th>
<th>#qubits</th>
<th>rigetti</th>
<th>ours</th>
<th>ibm</th>
<th>ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>2q qft</td>
<td>2</td>
<td>18</td>
<td>16</td>
<td>11%</td>
<td>12-42</td>
</tr>
<tr>
<td>3q qft</td>
<td>5</td>
<td>118</td>
<td>92</td>
<td>22%</td>
<td>57-73</td>
</tr>
<tr>
<td>ccnot</td>
<td>3</td>
<td>33</td>
<td>15</td>
<td>54%</td>
<td>18</td>
</tr>
<tr>
<td>4q add</td>
<td>8</td>
<td>197</td>
<td>132</td>
<td>33%</td>
<td>116-131</td>
</tr>
<tr>
<td>8q add</td>
<td>16</td>
<td>474</td>
<td>312</td>
<td>34%</td>
<td>272-299</td>
</tr>
<tr>
<td>mc3x</td>
<td>4</td>
<td>90</td>
<td>76</td>
<td>15%</td>
<td>46-102</td>
</tr>
<tr>
<td>mc4x</td>
<td>5</td>
<td>195</td>
<td>164</td>
<td>16%</td>
<td>94-150</td>
</tr>
<tr>
<td>2q grover</td>
<td>2</td>
<td>15</td>
<td>8</td>
<td>46%</td>
<td>16-27</td>
</tr>
<tr>
<td>bv</td>
<td>4</td>
<td>21</td>
<td>11</td>
<td>47%</td>
<td>22</td>
</tr>
</tbody>
</table>

selected results

applications

quantum linear solvers and optimization algorithms
potential quantum applications

- **HHL-type quantum linear solver**

  \[
  \min_{\theta} x^\dagger M_\theta x_
  \]

  \[
  \text{s.t. } A_\theta x_\theta = b_\theta
  \]

  Find \( x^\dagger M x \)  \( \text{s.t. } Ax = b \)

  - sparse matrices \( O(\log(N)\kappa^2/\epsilon) \)  \( \text{polylog}(1/\epsilon) \)  \( \text{polylog}(1/\epsilon) \)
    
    \[ \text{[Harrow, Hassidim, Lloyd 2009]} \]
    \[ \text{[Childs, Kothari, Somma 2017]} \]
  
  - dense matrices \( O(\sqrt{N}\log(N)\kappa^2/\epsilon) \)
    
    \[ \text{[Wossnig et al. 2018]} \]

- **applications**

  - linear differential equations [Berry 2010, Xin et al. 2018]
  - Poisson equation [Cao et al. 2013, Montanaro 2015]
  - principal component analysis [Lloyd et al. 2014]
  - data fitting [Wiebe et al. 2012]
caveats

- you don’t get the solution vector $x$ but a scalar value $x^\dagger M x$
- circuits are impractical for near-future quantum computers
- Recent step-by-step HHL algorithm walk-through by Morrell and Wong (08/2021):
  
  “[…] due to the imperfection and noise in a real quantum computer (ibmq_santiago), the hardware execution of the same circuit (for a 2x2 matrix) does not give satisfactory result”

arXiv:2108.09004

HHL simulation with Qiskit: 2x2 matrices, w/o noise

Experiments and Results

Experiments with the VQLS simulations use the same matrices as the HHL experiments, cf. Appendix D.

Experiments of size $2\times2$
The results of all $2\times2$ matrix tests can be found in Figure 4.10a. The runtime was mostly around 2.5 seconds with only two exceptions, $2.j$ and $2.i$ taking longer and one taking shorter, $2.l$. Fidelity results can be seen in Figure 4.10b, the results show a near perfect fidelity for all matrices, with the exception of $2.l$ which is very low.

(a) Runtimes for $2\times2$ matrices.
(b) Fidelity for $2\times2$ matrices.

Figure 4.10: Experiments with $2\times2$ matrices, state vector simulations.

Experiments of size $4\times4$
In experiments with the $4\times4$ matrices we have varied timer results, as can be seen in the graph in Figure 4.11a. Timings were between 10 and 20 seconds for most tests but for the denser random matrices $4.i$ and $4.j$ the timing was up to 52 and 81 seconds respectively. The fidelity, as shown in Figure 4.11b,

(a) Runtimes for $4\times4$ matrices.
(b) Fidelity for $4\times4$ matrices.

Figure 4.11: Experiments with $4\times4$ matrices, state vector simulations.

Experiments of size $8\times8$
The VQLS simulation can handle more complex $8\times8$ matrices but as we mainly wanted to compare the results to HHL it seemed redundant to test more matrices just for VQLS. The timing results are seen in Figure 4.12a and the fidelity in Figure 4.12b. Timings are considerably longer than for the $4\times4$ case with most runs being over 500 seconds long. The fidelity varies considerably but like in HHL.

HHL simulation with Qiskit: 2x2 matrices, with noise

Figure 4.13: Experiments with 2x2 matrices, noisy simulations.
(a) Runtimes for 2x2 matrices with noise.
(b) Fidelity for 2x2 matrices with noise.

Figure 4.14: Experiments with 4x4 matrices, noisy simulations.
(a) Runtimes for 4x4 matrices with noise.
(b) Fidelity for 4x4 matrices with noise.

The runtime was around 600 seconds, while most cases were around 300 seconds. Fidelity was worse still than in the smaller sizes though, as the highest value doesn't reach 0.16, see Figure 4.15b. These fidelity results are on such a small scale as well that they display no structure or are counter intuitive. For example from 8.\(f\) to 8.\(h\) there is an increase in fidelity, while the condition number increases from 10 to 1000, with sparsity unchanged.

Noisy Simulation of VQLS Analysis

The runtime of the VQLS noisy simulations showed results in the 4x4 to take the longest. This could relate to the ansatz or it may be due to the high number of shots needed to converge. In general, the fidelity results for all experiments was poor, with not even the identity matrix showing a fidelity of 1 in any test. It is hard to improve these results by tuning parameters as unlike HHL, VQLS does not rely on approximations like the QPE. We will look at parameter tuning in the next chapter.

HHL simulation with Qiskit: 4x4 matrices, w/o noise

Experiments and Results

Experiments with the VQLS simulations use the same matrices as the HHL experiments, cf. Appendix D.

Experiments of size $2 \times 2$
The results of all $2 \times 2$ matrix tests can be found in Figure 4.10a. The runtime was mostly around 2.5 seconds with only two exceptions, $2.j$ and $2.i$ taking longer and one taking shorter, $2.l$. Fidelity results can be seen in Figure 4.10b, the results show a near perfect fidelity for all matrices, with the exception of $2.l$ which is very low.

Figure 4.10: Experiments with $2 \times 2$ matrices, state vector simulations.

Experiments of size $4 \times 4$
In experiments with the $4 \times 4$ matrices we have varied timeresults, as can be seen in the graph in Figure 4.11a. Timings were between 10 and 20 seconds for most tests but for the denserandom matrices $4.i$ and $4.j$ the timing when up to 52 and 81 seconds respectively. The fidelity, as shown in Figure 4.11b,

(a) Runtimes for $4 \times 4$ matrices.
(b) Fidelity for $4 \times 4$ matrices.

Figure 4.11: Experiments with $4 \times 4$ matrices, state vector simulations.

Experiments of size $8 \times 8$
The VQLS simulation can handle more complex $8 \times 8$ matrices but as we mainly wanted to compare the results to HHL it seemed redundant to test more matrices just for VQLS. The timing results are seen in Figure 4.12a and the fidelity in Figure 4.12b. Timings are considerably longer than for the $4 \times 4$ case with most runs being over 500 seconds long. The fidelity varies considerably but like in HHL $8.a$ and $8.b$.

HHL simulation with Qiskit: 4x4 matrices, with noise

Experiments and Results

(a) Runtimes for 2x2 matrices with noise.
(b) Fidelity for 2x2 matrices with noise.

Figure 4.13: Experiments with 2x2 matrices, noisy simulations.

(a) Runtimes for 4x4 matrices with noise.
(b) Fidelity for 4x4 matrices with noise.

Figure 4.14: Experiments with 4x4 matrices, noisy simulations.

runtime was around 600 seconds, while most cases were around 300 seconds. Fidelity was worse still than in the smaller sizes though, as the highest value doesn't reach 0.16, see Figure 4.15b. These fidelity results are on such a small scale as well that they display no structure or are counter intuitive.

For example from $8.\mathbf{f}$ to $8.\mathbf{h}$ there is an increase in fidelity, while the condition number increases from 10 to 1000, with sparsity unchanged.

Noisy Simulation of VQLS Analysis

The runtime of the VQLS noisy simulations showed results in the 4x4 to take the longest. This could relate to the ansatz or it may be due to the high number of shots needed to converge. In general, the fidelity results for all experiments was poor, with not even the identity matrix showing a fidelity of 1 in any test. It is hard to improve these results by tuning parameters as unlike HHL, VQLS does not rely on approximations like the QPE. We will look at parameter tuning in the next chapter.

HHL simulation with Qiskit: 8x8 matrices, w/o noise

HHL simulation with Qiskit: 8x8 matrices, with noise

potential *near-future* quantum applications in SciComp

- **hybrid quantum-classical algorithms**
  - quantum approximate optimization algorithm (QAOA) [Farhi et al. 2014]
  - quantum alternating operator ansatz (QAOA) [Hadfield et al. 2017]
  - variational quantum eigensolver (VQE) [Peruzzo et al. 2014]
  - variational quantum linear solver (VQLS) for sparse matrices [Bravo-Prieto et al. 2019 & Xu et al. 2019]
QAOA workflow

truss structure optimization
3-truss structure

options: $2^{\#\text{trusses} \times \#\text{areas}} = 512$

preliminary results using Rigetti’s simulator

preliminary results using Rigetti’s Aspen-9 processor

only 9 out of 64 options are valid and the exclusion criterion is sensitive to noise

summary

- applied quantum computing is fun
- full-quantum algorithms are not near-future
- hybrid quantum-classical algorithms also have their subtleties
- nonetheless, it’s time for application-driven practical quantum computing

Thank you for your attention!